UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Sistema de análisis de rentabilidad de activos inmobiliarios

Javier Holgueras Crespo

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema de análisis de rentabilidad de activos inmobiliarios

Autor: Javier Holgueras Crespo
Tutor: Eduardo Cermeño Mediavilla

julio 2020

Javier Holgueras Crespo
Sistema de análisis de rentabilidad de activos inmobiliarios

Javier Holgueras Crespo
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia, amigos y tutores

Agradecimientos

A mi madre por todo el apoyo que me ha dado desde que nací ayudándome a seguir estudiando.

A todos mis amigos de la infancia y de la universidad por estar apoyándome y ayudándome en los mementos difíciles de la carrera.

A todos profesores del Grado de Ingeniería Informática de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid por haberme formado en esta gran carrera.

A todos los desarrolladores que de manera desinteresada ofrecen sus productos de manera gratuita y libre.

A mi tutor Eduardo y por proponerme este TFG y por toda su ayuda.

Resumen

Este proyecto se ha desarrollado con la idea de evaluar la rentabilidad de la inversión en el sector inmobiliario. El sector inmobiliario es un sector económico muy importante en España, Europa y el mundo dando trabajo a 2,4 millones de personas en 2017. En los últimos años, este sector ha sufrido un crecimiento muy importante debido al aumento del interés en comprar inmuebles como inversión invertir y no como vivienda. En cuanto a la inversión en inmuebles, una de las principales métricas para la renta es la rentabilidad potencial que se define como las ganancias que se esperan obtener. Por esto, es interesante desarrollar un sistema web que provea a los usuarios información de la rentabilidad potencial en diferentes zonas como provincias, municipios o barrios.

En la actualidad no hemos encontrado ninguna aplicación que permita obtener la rentabilidad inmobiliaria en diferentes zonas de forma sencilla y actualizada. Solo se han encontrado informes de rentabilidad que suelen salir cada 6 meses e informan únicamente del rendimiento de las provincias y municipios más importantes y aplicaciones de compra/venta de inmuebles cuyo objetivo es la venta y para obtener la rentabilidad hace falta hacer los cálculos manualmente. Por esta razón, este proyecto es novedoso e interesante.

Para el desarrollo de este sistema, se ha optado por utilizar un servicio web debido a su sencillez, rapidez y alcance. Para ello, se han utilizado herramientas muy conocidas para los desarrolladores para agilizar y facilitar en lo mayor posible el desarrollo como Django, Python, HTML, Google Maps o Scrapy. Para obtener la información de los inmuebles se ha optado por seguir dos frentes diferentes. En un frente ha usado la API REST que ofrece Idealista para obtener los inmuebles de su base de datos y para eliminar las limitaciones de su servicio gratuito de 100 peticiones mensuales se ha optado por desarrollar, en un segundo frente, un *crawler* mediante el *framework* Scrapy para extraer datos de su página web.

Como resultado, la aplicación proporciona la información que el usuario necesita para invertir como el rendimiento potencial, el precio medio de venta y de alquiler y el tamaño medio de los inmuebles. El proceso de desarrollo ha tenido algunas complicaciones debido sobre todo a la falta de documentación o a errores en ella pero finalmente se ha conseguido desarrollarla según lo previsto.

Palabras clave

Inversión, Propiedades, Inmuebles, Idealista, API, Python, Django, Web, Scrapy

Abstract

This project has been developed with the idea of evaluating the profitability of investment in the real estate sector. The real estate sector is a very important economic sector in Spain, Europe and the world, employing 2.4 million people in 2017. In recent years, this sector has experienced a significant growth due to the increased interest in buying real estate as an investment and not as a home. As for real estate investment, one of the main metrics for income is the potential profitability that is defined as the expected profits. For this reason, it is interesting to develop a web system that provides users with information on potential profitability in different areas such as provinces, municipalities or neighborhoods.

At present we have not found any application that allows obtaining real estate profitability in different areas easily and updated. We have only found profitability reports, that usually come out every 6 months and only report the performance of the most important provinces and municipalities, and real estate purchase/sale applications whose objective is the sale and to obtain profitability it is necessary to do the calculations manually. For this reason, this project is novel and interesting.

For the development of this system, it has been chosen to use a web service due to its simplicity, speed and scope. To do this, well-known tools for developers have been used to speed up and facilitate development as much as possible such as Django, Python, HTML, Google Maps or Scrapy. To obtain the information about the properties, we have chosen to follow two different fronts. On one front, it has used the REST API that Idealista offers to obtain the properties of its database and to remove the limitations of its free service of 100 monthly requests, it has been chosen to develop, on a second front, a crawler using the Scrapy framework to extract data from its website.

As a result, the application provides the information that the user needs to invest such as the potential profitability, the average sale and rental price and the average size of the properties. The development process has had some complications, mainly due to the lack of documentation or errors in it, but finally it has been developed as planned.

Keywords

Investment, Properties, Estates, Idealista, API, Python, Django, Web, Scrapy

Índice

Listas

Lista de algoritmos

Lista de códigos

Lista de cuadros

Lista de ecuaciones

Lista de figuras

Lista de tablas

Lista de cuadros

1

Introducción

1.1. Motivación

Este proyecto surge con la idea de desarrollar un sistema web que permita evaluar la rentabilidad potencial de la inversión en el sector inmobiliario. En 2017, el sector inmobiliario movió en Europa 240.000 millones de euros dando trabajo a 1,3 millones de empresas y 2,4 millones de trabajadores [1]. En cuanto a España, el sector inmobiliario es uno de los principales sectores económicos. En el año 2017, este sector representaba el 5 % del PIB (Producto Interior Bruto) del país [2] y como ejemplo, solamente el mercado de la vivienda movió 74.000 millones de euros [3] ese mismo año.

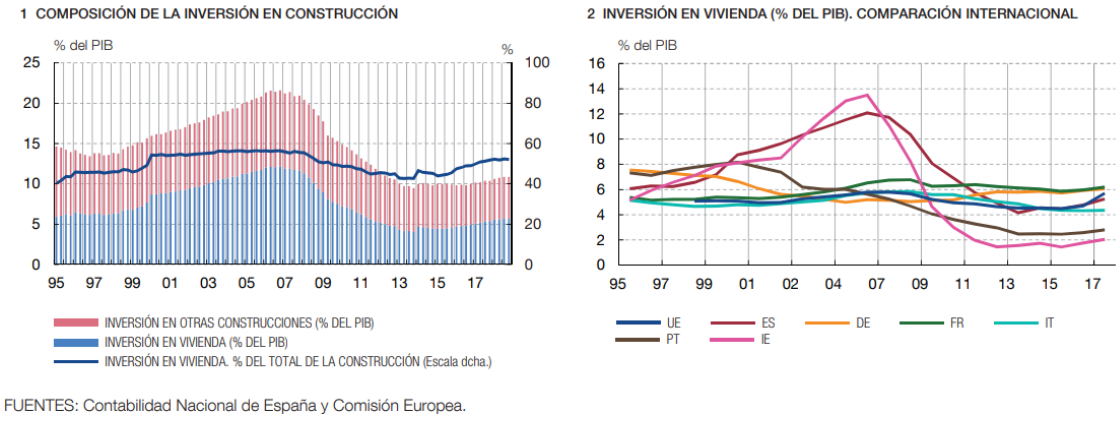

Figura 1.1: Evolución del sector inmobiliario en los principales países europeos [2]

Este sector inmobiliario se espera que aumente ligeramente en los próximos años [4] pero más lentamente comparado con el “boom"de este sector [5]. En los últimos años, las empresas especializadas en este sector han sufrido un crecimiento muy importante. A la cabeza se encuentran las inmobiliarias que han crecido un 70 % desde el año 2013 gracias a una reforma de la ley y sobre todo al interés de grandes y pequeños inversores [6]. Este interés radica en que cada vez más personas piensan en comprar una vivienda como inversión en lugar de para vivir causado principalmente por el aumento de la demanda de alquileres entre los jóvenes [7]. Por tanto, es interesante disponer de un sistema web para que cualquiera, sea particular o profesional, pueda acceder y consultar la rentabilidad potencial

de la inversión en lugar de ver el tradicional precio de los inmuebles.

La rentabilidad es uno de los 4 principales factores de una inversión y se define como el rendimiento o cantidad de dinero que se espera obtener potencialmente en una inversión que es “el acto de postergar el beneficio inmediato del bien invertido por la promesa de un beneficio futuro más o menos probable" [8]. La rentabilidad es el factor a tener en cuenta a la hora de invertir en un inmueble especialmente para su renta o alquiler que es la opción más común de inversión inmobiliaria en España entre particulares. La rentabilidad del alquiler se calcula mediante la división del precio del alquiler anual entre el precio de venta del inmueble. Por la mayor popularidad del arrendamiento, parece relevante enfocar el sistema en la evaluación del rendimiento potencial del alquiler en diferentes zonas.

1.2. Objetivos

El principal objetivo de este proyecto es la creación de una aplicación web sencilla y fácil de utilizar que permita buscar inmuebles por zonas definidas en diferentes niveles (provincias, municipios, distritos y barrios) para evaluar la rentabilidad potencial de esas zonas así junto con estadísticas adicionales como el precio medio de venta, el precio medio de alquiler y el tamaño medio de los inmuebles. Adicionalmente, la aplicación debería permitir el filtrado mediante características físicas de los inmuebles como su tamaño y mediante la evolución de la población donde se ubica. Otro objetivo es utilizar fuentes de información disponibles actualmente en internet que provean de una gran cantidad de datos relevantes para la aplicación como bases de datos de inmuebles y bases de datos de población.

1.3. Estructura del documento

- **Introducción:** En este capítulo se introduce al lector en la temática del proyecto describiendo el problema que debe resolver exponiendo las motivaciones que han llevado a su realización y los objetivos que pretende cumplir.
- **Estado del Arte:** Este capítulo se dedica a presentar al lector el mundo donde la aplicación coexistirá exponiendo los métodos actuales de obtención de las estadísticas mostrando aplicaciones similares.
- **Análisis y diseño**: Dentro de este capítulo, se detalla el análisis realizado al problema exponiendo los subsistemas y los requisitos de la aplicación tanto funcionales como no funcionales, así como también el ámbito y el alcance del proyecto. En cuanto al diseño, se explica el patrón de diseño elegido, los modelos tanto de la aplicación como de la base de datos que se necesitan, los diagramas de secuencia y las diferentes pantallas de las que se compone la aplicación.
- **Desarrollo:** En este capítulo se detallan las diferentes tecnologías, herramientas y servicios que se han usado para el desarrollo del producto explicando sus ventajas y desventajas y su utilización en este proyecto.
- **Pruebas y resultados:** Este capítulo está enfocado en mostrar las diferentes pruebas que se han realizado para verificar el correcto funcionamiento de la aplicación y los resultados que se han obtenido explicando, para ambos, los métodos que se han utilizado.
- **Conclusiones y trabajo futuro:** Este capítulo se enfoca en extraer las conclusiones tanto del desarrollo como de la aplicación exponiendo posibles mejoras o cambios que se podrían realizar en un futuro.

2

Estado del Arte

Dado el tamaño y la importancia del sector inmobiliario, no solo en España sino también en Europa y en el resto del mundo y el aumento de la inversión en este sector, no es de extrañar que haya muchas iniciativas para proporcionar información al respecto. Actualmente existen dos fuentes principales de información.

La primera fuente de información para obtener datos inmobiliarios son los informes de rentabilidad. Estos informes proporcionan información acerca del rendimiento inmobiliario potencial de ciertas áreas. Las áreas que suelen aparecer en los informes son las provincias y las áreas más populares del país que coinciden con las grandes capitales como Madrid, Barcelona, Sevilla o Valencia y sus alrededores. Los informes más comunes son los de las provincias y suelen salir cada 6 meses en los mejores casos. Los informes que muestran el rendimiento inmobiliario por municipios son más escasos y muestran solo los principales municipios de cada provincia y/o el municipio con mayor rendimiento (puede no ser uno de los principales). Estos informes suelen salir cada más tiempo estando la media en el año. También hay otro tipo de informes mucho menos comunes que se enfocan en los distritos y barrios de las principales ciudades como Madrid o Barcelona. Estos informes tienen menos alcance que los demás por que se publican en prensa local o en prensa nacional, pero en la sección provincial. Estos últimos informes tienen el mismo periodo que los municipales y salen más o menos cada año. La ventaja de los informes es su simplicidad ya que proporcionan la información muy resumida, pero en la simplicidad también radica una de sus desventajas ya que, si se desea conocer el rendimiento inmobiliario de una zona no muy popular, es muy probable que no exista o su publicación sea muy antigua y desfasada. El periodo es su otra desventaja. Al estar publicados cada cierto tiempo no reflejan correctamente el sector salvo que se consulte no muchos días después de su publicación. A continuación, se muestran algunas aplicaciones que realizan y/o muestran estos informes:

- **Idealista data:** [9] En esta aplicación web puede consultarse el precio del metro cuadrado de los inmuebles en cada provincia, así como también ofrece una comparación con los anteriores valores y con el global de España. Estos datos se actualizan cada trimestre del año y ofrecen datos de los últimos 3 años.
- **Idealista news:** [10] En esta sección del popular portal de compra/venta de inmuebles, Idealista, se pueden encontrar informes que muestran el rendimiento de los inmuebles en cada provincia y tipo de inmueble cada trimestre y ofrecen un resumen muy general del rendimiento [11]. También ofrecen informes más detallados incluyendo los distritos de las principales ciudades así como de los municipios de las principales provincias. Estos

últimos informes no se ofrecen con tanta regularidad como los demás, pero son mucho más detallados. [12]. Este portal además ofrece datos como el porcentaje de descuento que pide la gente a la hora de hacer una contraoferta por un inmueble [13] y noticias relacionadas con el mundo inmobiliario.

- **Prensa generalista:** En algunos periódicos generalistas como El Confidencial [14] [15] o Expansión [16] ofrecen de vez en cuando informes que han calculado sus expertos mediante diferentes tecnologías como datacrawling o informes de terceros. En la figura 2.1 se puede ver un mapa del rendimiento inmobiliario del alquiler en las provincias y los municipios principales de España en el primer trimestre de 2019 del periódico Expansión.

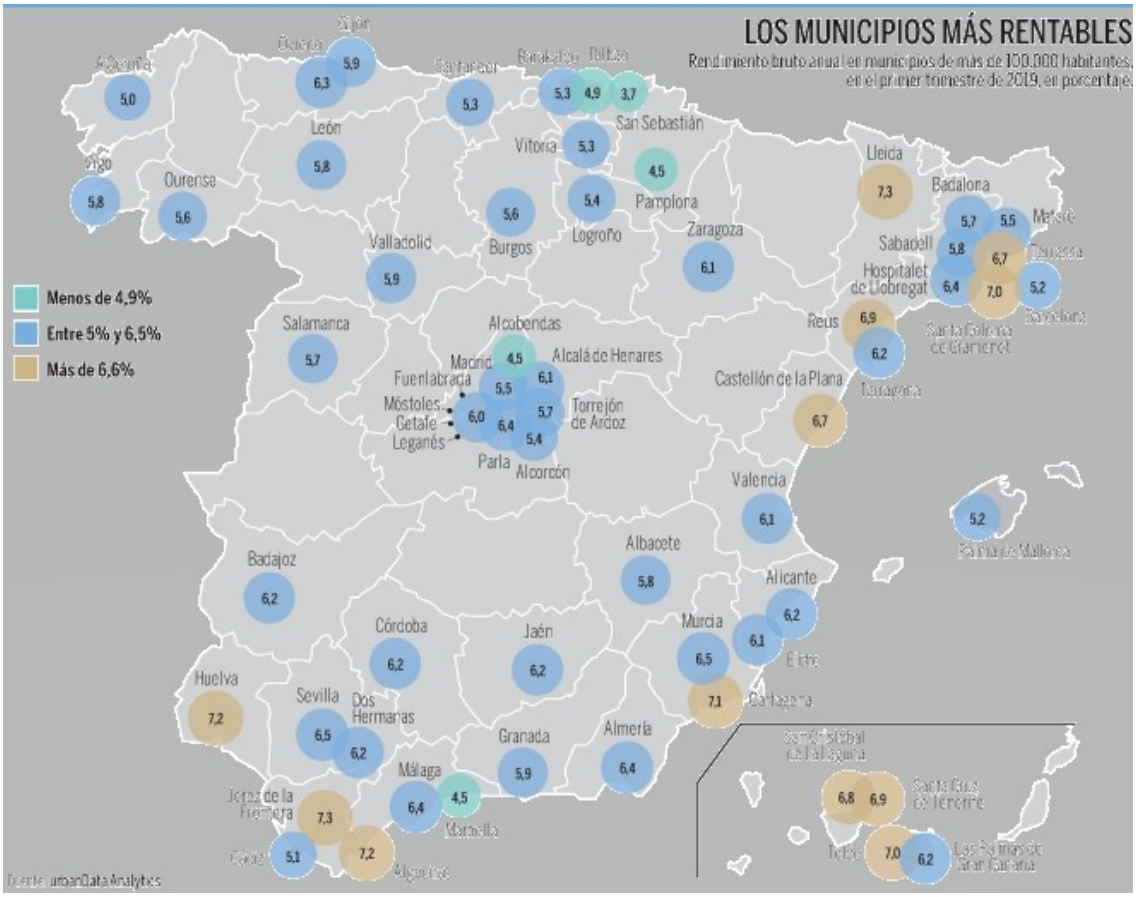

Figura 2.1: Rendimiento inmobiliario del alquiler de los principales municipios [16]

La otra fuente principal de información son los portales inmobiliarios de compra/venta de inmuebles. Estos portales, que en su gran mayoría están disponibles online a todo el mundo, proporcionan una manera muy sencilla de consultar inmuebles mediante diferentes filtros como la ubicación o las características físicas de los inmuebles. Su principal ventaja es que se muestran los inmuebles que actualmente están en venta y con ese precio con lo que continuamente están actualizados. Su principal y gran desventaja es su objetivo. El objetivo de estos portales inmobiliarios es la venta de inmuebles y solo proporcionan información para ello como el precio, el tamaño, las características, la descripción, las fotos, etc. En ningún momento proporcionan información sobre el rendimiento potencial del alquiler. Para realizar el cálculo del rendimiento, el usuario debe calcularlo manualmente mediante el precio de venta por metro cuadrado y el precio de alquiler anual por metro cuadrado y para ello debe analizar todos los inmuebles que estos portales ofrecen sobre una zona en concreto. Para comparar las

rentabilidades de diferentes zonas el usuario debe realizar los pasos anteriores para cada una de las zonas que desea comparar de manera que este método no es eficiente ni recomendable. Algunos de los principales portales inmobiliarios son los siguientes:

- **Idealista:** [17] Este es el portal inmobiliario líder en compra/venta de inmuebles en España. Permite buscar todo tipo de inmuebles basándose en diferentes filtros como la ubicación, el tamaño o el precio. Permite obtener rápidamente el valor de los inmuebles, así como los alquileres pero no ofrece datos acerca de la rentabilidad. Este portal además ofrece datos relevantes como el precio de venta medio por metro cuadrado de inmuebles en diferentes zonas de España, ya sean distritos de una gran ciudad o un municipio de pocos habitantes, desde que el portal fue creado. [18]
- **Fotocasa:** [19] Este portal inmobiliario de compra/venta de inmuebles es uno de los principales portales de España. Este portal, al igual que Idealista posee un blog donde suben noticias, artículos, etc. relacionados con la vivienda y los inmuebles. De vez en cuando suben informes de rendimiento inmobiliario [20] [21] donde exponen las provincias, los municipios, los distritos y los barrios más rentables pero solo ofrecen los datos completos de las provincias. Este blog es un complemento de su aplicación y su principal objetivo es la venta de inmuebles de igual forma que Idealista.

Ningún informe o portal inmobiliario ofrece la rentabilidad de la renta de manera sencilla y actualizada. En cambio, en este proyecto, se propone que haya una aplicación similar a las anteriores pero que en lugar de estar focalizada en la compra/venta de inmuebles este centrada en el concepto de rentabilidad lo que hace este proyecto novedoso e interesante.

3

Análisis y Diseño

Tras haber analizado el tamaño y la importancia del sector inmobiliario y su gran evolución y el aumento del interés en la inversión en inmuebles creemos que podemos ofrecer un sistema mejor que los actuales métodos de obtención de la información con una solución para conocer la rentabilidad potencial de las inversiones inmobiliarias de forma interactiva. De todas las aplicaciones similares que existen hoy en día nos ha llamado especialmente la atención los portales inmobiliarios de compra/-venta de inmuebles. Estos portales poseen en la actualidad una enorme base de datos de inmuebles que servirían como fuente de obtención de los datos para evaluar la rentabilidad por zonas. De los dos principales pórtales que existen en España, nos ha llamado la atención Idealista. Idealista no es solamente un portal inmobiliario, es además un noticiario especializado en temas inmobiliarios y un proveedor de estadísticas. Analizando todos los servicios que ofrecen se ha encontrado un servicio que ofrece en forma de API del cual se pueden obtener información básica de los inmuebles haciendo peticiones con una serie de filtros para filtrar que tipo de inmueble se desea, con qué características y donde se ubica. Viendo todo lo que ofrece, se ha optado por usar este servicio para obtener los datos de los inmuebles.

Otra información importante que se podría ofrecer a los usuarios es la evolución de la población en las diferentes zonas para hacerse una idea del crecimiento futuro del lugar como información extra para la inversión. Para obtener esta información se buscó en el INE o Instituto Nacional de Estadística. El INE ofrece de manera gratuita y abierta una serie de estadísticas de interés para muchas personas. El INE ofrece la información de la población de municipios y provincias cada año y además ofrece una API para obtener esta información fácilmente por lo que al final se eligió proveedor de estadísticas de población.

3.1. Ámbito y alcance de la aplicación

El principal objetivo del sistema es integrar en una única aplicación toda la obtención de las características de los inmuebles para posteriormente realizar una serie de cálculos que concluirán en unos resultados de estadísticas inmobiliarias como el rendimiento del alquiler, el precio medio de venta o de

alquiler o el tamaño medio de los inmuebles en sus respectivas zonas. En particular, el sistema debe proporcionar una serie de filtros para poder filtrar los inmuebles basados en características propias o sobre la población del municipio donde se encuentran. El objetivo de esta aplicación no es mostrar estos inmuebles según sus filtros, sino proporcionar estadísticas que no están disponibles en ninguna aplicación. El sistema no proporcionará ningún método para añadir inmuebles como si se tratase de un portal de compra/venta de inmuebles donde se pueden publicar anuncios. El sistema solo mostrará recomendaciones de inmuebles y en ningún caso mostrará información sobre el anuncio del inmueble aparte de su precio, tamaño y rendimiento, sino que redirigirá al usuario al correspondiente portal inmobiliario donde fue encontrado.

3.2. Subsistemas

Los diferentes subsistemas que componen la aplicación están diseñados de tal forma que cada uno de ellos realiza funciones muy concretas y relacionadas entre sí para poder desarrollar y diseñar con más facilidad y comodidad la aplicación pudiendo separar los requisitos según el conjunto de su funcionalidad. En total hay 5 subsistemas y cada uno de ellos agrupa funcionalidades comunes entre sí. Se ha decido separar el subsistema de búsqueda y subsistema de recomendación en dos porque, aunque su funcionalidad es muy parecida y común, en cada uno hay un extra en cada funcionalidad que el otro no tiene. Los 5 subsistemas son los siguientes:

Subsistema de búsqueda

Este subsistema es el encargado de mostrar las principales estadísticas de los inmuebles por zonas basándose en diferentes filtros proporcionados por el usuario. El subsistema pedirá al usuario una serie de filtros básicos para poder analizar los inmuebles como la localización que desea analizar y el tipo de inmueble en el que está interesado pudiendo elegir entre viviendas, locales, garajes y oficinas. El sistema analizará los inmuebles encontrados y mostrará la información de las estadísticas según el nivel de búsqueda que el usuario ha elegido de las 4 opciones que existen (a nivel provincial, municipal, distritos y barrios).

Subsistema de recomendación

Se encargará de mostrar al usuario tanto las estadísticas básicas de los inmuebles como una pequeña recomendación de los mejores inmuebles que el usuario podrá adquirir basándose en los filtros que ha indicado. Los filtros serán parecidos a los filtros del subsistema de búsqueda con el añadido de que podrá filtrar los inmuebles por su precio. El sistema no mostrará los diferentes niveles de búsqueda ya que solo mostrará el área circular que el usuario ha indicado ignorando los diferentes municipios, distritos, barrios... que hay en su interior. De esta forma este subsistema mostrará menos

información estadística que el subsistema de búsqueda para enfocarse en la recomendación.

Subsistema de Idealista

Este subsistema es el encargado de obtener todos los inmuebles del portal líder en compra/venta de inmuebles, Idealista. Se encargará de obtener los inmuebles basándose en algunas características que poseen obtenidas de los filtros de los dos subsistemas anteriores. También se encargará de descargar los inmuebles de Idealista mediante su API o de obtenerlos de la base de datos creada por el *crawler*. Este subsistema se encargará internamente de la gestión de las credenciales de acceso a la API y su renovación. Este subsistema se ha creado porque es la parte común más grande que compone los subsistemas de búsqueda y recomendación y agrupa toda la obtención de los inmuebles.

Subsistema de Población

El subsistema de población es el encargado de suministrar diferentes estadísticas de población según la localización aportada. Se encarga de obtener todos las provincias y pueblos de España de la base de datos del INE, obteniendo así los códigos que el Instituto Nacional de Estadísticas ha proporcionado a cada uno como identificación. Con estos datos, se encargará de proporcionar las cifras de población de un municipio o provincia en un periodo de tiempo. Esta parte también es común en los subsistemas de búsqueda y de recomendación y agrupa la funcionalidad de obtención de los datos de población.

Subsistema de *scraping*

El subsistema de *scraping* es el subsistema encargado de obtener todos los posibles datos sobre los diferentes tipos de inmuebles mediante el *scraping* de las páginas web de los principales portales inmobiliarios. Este subsistema se encargará de navegar por las diferentes páginas web obteniendo la información del inmueble y convirtiéndola en los tipos de datos que maneja la aplicación. Posteriormente se encargará de guardar esa información en la base de datos para su futura consulta, obtención y análisis. Este subsistema es especial porque se realiza separado de la aplicación y los datos que recoge serán usados por el subsistema de Idealista.

3.3. Requisitos

Los requisitos se han divido en requisitos funcionales, que definen funciones del sistema, y en requisitos no funcionales, que definen cualidades del sistema. Los requisitos funcionales están agrupados por los diferentes subsistemas que se han definido.

3.3.1. Requisitos funcionales

Subsistema de búsqueda

RF-1.– La aplicación permitirá buscar la zona más rentable (zona con la diferencia entre el precio del alquiler medio anual entre el precio de venta medio más alto).

RF-2.– El sistema permitirá buscar en un área mediante una búsqueda por palabras como si de un buscador se tratase

RF-3.– El sistema debe hacer la comprobación de que el lugar indicado existe en el territorio español y si no es el caso deberá informar de dicho error adecuadamente.

RF-4.– La aplicación permitirá seleccionar un radio de búsqueda en metros de tal forma que se buscarán todos los inmuebles que se encuentren dentro del área del circulo definido por el centro de la dirección y su radio.

RF-5.– El sistema permitirá seleccionar el tipo de inmueble que se desea filtrar de entre los siguientes:

- Vivienda
- Local
- Oficina
- Garaje

RF-6.– El sistema permitirá filtrar los resultados según una serie de filtros comunes y una serie de filtros propios de cada tipo de inmueble.
Los filtros comunes son los siguientes:

- Tipo de inmueble
- Nivel de búsqueda
- Lugar de búsqueda
- Radio de búsqueda
- Si el inmueble es propiedad de una entidad bancaria

Los filtros propios de cada inmueble son los siguientes:

- Viviendas:
  - Tamaño mínimo y máximo en metros cuadrados
  - Tipo de vivienda a elegir entre los siguientes:
    - Piso o apartamento
    - Ático
    - Dúplex
    - Estudio
    - Chalé
    - Casa de campo
  - Número de habitaciones a elegir entre los siguientes pudiendo escoger varios a la vez:
    - 0 (estudio)
    - 1
    - 2
    - 3

        - 4 o más
    - Número de cuartos de baño a elegir entre los siguientes pudiendo escoger varios a la vez:
        - 0
        - 1
        - 2
        - 3 o más
    - Estado de la vivienda a elegir entre los siguientes:
        - Nueva construcción
        - En buen estado
        - Para reformar
    - Si la vivienda está amueblada a elegir entre los siguientes:
        - Vivienda amueblada
        - Cocina y vivienda amueblada
    - Si la vivienda tiene aire acondicionado o no
    - Si la vivienda tiene armarios empotrados o no
    - Si la vivienda o el edificio tiene ascensor o no
    - Si la vivienda da al exterior o no
    - Si la vivienda tiene o viene con plaza de garaje incluida o no
    - Si la vivienda o el edificio tiene piscina o no
    - Si la vivienda tiene terraza o no
    - Si la vivienda viene con trastero o no
    - Si la vivienda tiene tendedero o no
- Locales:
    - Tamaño mínimo y máximo en metros cuadrados
    - Tipo de local a elegir entre los siguientes:
        - Local
        - Edificio industrial (Nave industrial)
    - Localización del local a elegir entre las siguientes:
        - En centro comercial
        - A nivel de calle
        - Entreplanta
        - Subterráneo
        - Otros
    - Si el local hace esquina o no
    - Si el local tiene aire acondicionado o no
    - Si el local tiene calefacción o no
    - Si el local tiene extractor o salida de humos o no
    - Si el local es transferible o no

- Oficinas:
  - Tamaño mínimo y máximo en metros cuadrados
  - Diseño de la oficina a elegir entre los siguientes:
    - Con tabiques
    - Abierto
  - Uso del edificio de la oficina a elegir entre los siguientes:
    - Exclusivo
    - Mixto
  - Si la oficina tiene agua caliente o no
  - Si la oficina tiene aire acondicionado o no
  - Si la oficina o el edificio tiene ascensor o no
  - Si la oficina tiene calefacción o no
  - Si la oficina tiene o viene con garaje incluido o no
  - Si la oficina tiene sistemas de seguridad y guardias o no
- Garajes:
  - Si el garaje es para motos o no
  - Si el garaje tiene puertas automáticas
  - Si el garaje tiene sistema de seguridad y guardias

RF-7.– La aplicación mostrará las diferentes zonas que se encuentran dentro del área indicada a la vez que mostrará el resumen del área en primer lugar. Las zonas que se mostrarán variarán según el nivel de búsqueda indicado mostrando los barrios, distritos, municipios y provincias y estarán ordenadas de mayor a menor rentabilidad potencial.

RF-8.– El sistema mostrará el tipo de zona y el nombre en los resultados junto con el rendimiento del alquiler (este debe destacar sobre el resto), el tamaño medio de los inmuebles, el precio medio de compra y el precio medio del alquiler tanto total como por metros cuadrados.

RF-9.– El sistema mostrará en cada zona la evolución de su población solamente a nivel de municipio o provincia. Si las zonas son barrios o distritos se mostrará la evolución de la población de su municipio. En el caso del área se indicará la suma de la población de los municipios que se encuentran dentro del área.

RF-10.– La aplicación mostrará una pequeña búsqueda rápida en cada zona y en el área donde se especificará el rendimiento potencial según diferentes criterios en cada tipo de inmueble:

- Viviendas: Se mostrará una tabla comparativa del rendimiento según su tamaño y estado. Para el tamaño se usarán los siguientes rangos en metros cuadrados: (20, 60), (60, 100), (100, 140), (140, 180), (180, 270), (270, infinito)

RF-11.– El sistema mostrará un resumen con los filtros aplicados en la búsqueda para una mejor comprensión de los resultados.

RF-12.– Los filtros no deben de aplicarse hasta que el usuario realice la acción de aplicarlos.

RF-13.– El sistema tendrá filtros de población para filtrar inmuebles según el incremento o decremento de la población en su ubicación (municipio o provincia).

RF-14.– El sistema mostrará en un mapa el área del circulo donde se están buscando los inmuebles con centro en la dirección y con radio igual a la distancia de búsqueda.

Subsistema de recomendación

RF-15.– El sistema de recomendación tendrá los mismos filtros de características que el subsistema de búsqueda al que se le añade un filtro para filtrar por el precio mínimo y máximo de los inmuebles y se quite el filtro de nivel de búsqueda.

RF-16.– La aplicación mostrará solo las recomendaciones para el área del circulo.

RF-17.– El sistema mostrará las mismas estadísticas que en el subsistema de búsqueda excepto la búsqueda rápida.

RF-18.– La aplicación mostrará los mejores 3 inmuebles para el total del área y para cada estadística de la búsqueda rápida del subsistema de búsqueda. Por ejemplo, en el caso de los inmuebles, se recomendará las mejores 3 viviendas para cada rango de tamaños y para cada estado del inmueble.

RF-19.– Para cada uno de los inmuebles recomendados, el sistema mostrará una pequeña imagen principal, un titulo adecuado (Como el tipo de inmueble y la dirección), el precio total de venta, el rendimiento con respecto a su franja y un enlace al portal inmobiliario del que procede.

RF-20.– La aplicación mostrará, al igual que con el subsistema de búsqueda, un resumen de los filtros aplicados en la búsqueda.

RF-21.– Los filtros no deben de aplicarse hasta que el usuario realice la acción de aplicarlos al igual que con el subsistema de búsqueda.

RF-22.– La aplicación mostrará un mapa con el área de igual forma que lo hace en el subsistema de búsqueda.

Subsistema de Idealista

RF-23.– El sistema permitirá comunicarse con la API (Application Programming Interface o interfaz de programación de aplicaciones en español) de Idealista para obtener los inmuebles.

RF-24.– La aplicación proporcionará todos los filtros que la API proporciona para cada uno de los tipos de inmuebles como también los filtros comunes.

RF-25.– El sistema gestionará de forma automática la renovación del *token* basado en clave y contraseña de forma que solo se tenga que especificar la clave y el secreto una vez.

RF-26.– El sistema gestionará la obtención de todos los inmuebles si estos se encuentran en diferentes páginas y devolverá una lista con todos los inmuebles (en forma de diccionario) que devuelve.

RF-27.– La aplicación se encargará de crear los objetos de inmuebles con el modelo de Django usando la lista de diccionarios que devuelve la API.

Subsistema de Población

RF-28.– El sistema obtendrá de la base de datos del INE el código o identificador de cada una de las provincias y municipios de España.

RF-29.– El sistema deberá proporcionar la población de una provincia o un municipio por diferentes métodos que son los siguientes:

- Últimos N datos: La aplicación proporcionará los últimos N datos que haya en la base de datos del INE y devolverá un diccionario con la población por cada año.
- Rango de fechas: La aplicación proporcionará los datos de la población entre un rango de fechas y devolverá un diccionario con la población por cada año.

Subsistema de *scraping*

RF-30.– La aplicación debe poder obtener todos los datos de los inmuebles mediante *scraping* de la página web de Idealista.

RF-31.– El sistema debe obtener todos los inmuebles proporcionando una provincia y una operación.

RF-32.– El sistema navegará por cada una de las páginas de la provincia.

RF-33.– La aplicación obtendrá todos los datos relevantes de cada uno de los tipos de inmuebles que existen en la web de Idealista y los guardará en una base de datos convirtiendo antes los datos obtenidos a los datos del modelo.

3.3.2. Requisitos no funcionales

RNF-1.– Interfaz sencilla y moderna: El sistema debe proporcionar una interfaz sencilla y agradable para la mayoría de los usuarios de hoy en día.

RNF-2.– Compatibilidad con los navegadores web más populares de escritorio (cualquier navegador basado en Gecko, Chromium o WebKit)

RNF-3.– Privacidad: El sistema solo ha de guardar los datos imprescindibles de los usuarios para preservar su privacidad.

RNF-4.– Sistema Operativo: El servidor ha de funcionar en los sistemas operativos de escritorio más populares (Windows NT, GNU/Linux y macOS)

RNF-5.– Pantalla: La aplicación ha de ser compatible con las resoluciones de pantalla más populares en formato horizontal.

RNF-6.– Idiomas: La aplicación ha de estar disponible en español y en inglés.

3.4. Patrón de diseño

El patrón de diseño que se ha decidido usar para esta aplicación es el patrón MVC (Modelo-Vista-Controlador). Este patrón de diseño es uno de los más populares y permite separar la aplicación en tres partes: el modelo donde se encuentra la lógica de la aplicación, la vista que es la parte que se muestra al usuario y el controlador que es la parte que accede a la lógica y la muestra en la vista conectando ambas partes.

Más concretamente, se ha usado el modelo MVT (Modelo-Vista-Template). Este modelo se diferencia del MVC en las plantillas o *templates*. En el modelo MVT se separa los datos y la funcionalidad en tres: la interfaz de usuario (plantilla), la lógica de negocio (modelo) y la parte que une las dos (vista). Este modelo es el que usa Django para crear páginas y servicios web.

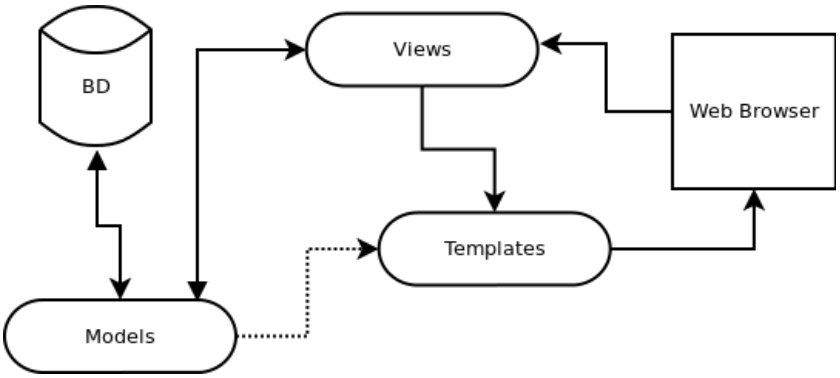

Figura 3.1: Modelo MVT

3.5. Arquitectura

Esta aplicación utiliza una arquitectura basada en páginas web que consta de dos partes diferenciadas. Es una estructura muy típica en ingeniería del software. Consta de una parte llamada *front-end* que es la parte que se encarga de mostrar los datos e interaccionar con los elementos y las acciones de los usuarios. La otra parte es la llamada *back-end* que es la que se encarga de procesar las peticiones del *front-end* y realizar las operaciones necesarias con el modelo para responder a la petición.

En este caso, como aplicación *back-end* se ha usado el lenguaje de programación Python con el *framework* Django. Este *framework* ofrece todas las características necesarias para el *back-end* proporcionando el modelo MVT. Como base de datos para el *framework* se ha usado el gestor de base de datos relacional PostgreSQL.

Como *front-end* se ha usado un navegador que muestra páginas HTML que el servidor envía al cliente junto con las hojas de estilo CSS, que aportan el diseño a la página, y algunos ficheros de *script* que permiten controlar algunos parámetros imposibles de hacer desde el *back-end*.

3.6. Modelos de base de datos

Los modelos que se han usado para el desarrollo de esta aplicación están creados con el *framework* Django. Django provee una manera sencilla de definir modelos mediante código Python dentro de la aplicación y Django se encarga de traducir el código para la creación de las tablas. Todos los modelos de Django que se han usado están en el anexo A y los campos que su valor depende de una opción se puede encontrar su opción en el anexo B.

A continuación, se describen para que sirven cada uno de los campos del modelo que se usa para guardar los datos que se obtienen mediante la API de Idealista. Este modelo es el modelo principal de la aplicación:

- property_code: Indica el identificador del inmueble en Idealista.

- title: Titulo del inmueble
- subtitle: Subtitulo del inmueble
- latitude: Latitud (coordenadas) del inmueble
- longitude: Longitud (coordenadas) del inmueble
- distance: Distancia desde la longitud y latitud indicados en la búsqueda hasta el inmueble.
- country: País al que pertenece el inmueble que puede tomar los valores indicados en el código B.2
- province: Provincia al que pertenece el inmueble
- region: Region al que pertenece el inmueble
- subregion: Subregión al que pertenece el inmueble
- municipality: Municipio al que pertenece el inmueble
- district: Distrito al que pertenece el inmueble (Puede ser nulo)
- neighborhood: Barrio al que pertenece el inmueble (Puede ser nulo)
- address: Dirección del inmueble
- floor: Piso en el que se encuentra el inmueble
- size: Tamaño en metros cuadrados del inmueble
- bathrooms: Baños que tiene el inmueble
- exterior: Indica si el inmueble es exterior
- state: Estado de conservación del inmueble que puede tomar los valores indicados en las opciones mostradas en el código B.4.
- has_video: Indica si el inmueble tiene un vídeo en su anuncio.
- num_photos: Indica el número de fotos del inmueble en el anuncio.
- operation: Operación del inmueble que puede tomar los valores indicados en el código B.3.
- price: Precio en euros del inmueble.
- show_address: Indica si el anunciante ha proporcionado la dirección exacta del inmueble.
- thumbnail: Foto de portada del anuncio.
- url: Dirección al anuncio en el portal de Idealista.
- typology: Indica la tipología del inmueble que puede variar según el tipo de inmueble que puede tomar los valores indicados en las opciones mostradas en el código B.5.
- subtypology: Subtipología del inmueble que variar con el tipo de inmueble y que puede tomar los valores indicados en las opciones mostradas en el código B.6.

3.7. Diagramas

En la figura 3.2 se muestra la secuencia de llamadas principales que ocurren al buscar inmuebles en la página web. En este caso, el usuario realiza una petición proporcionando los filtros, el sistema procesa y valida el formulario. Después procesa la dirección obteniendo sus coordenadas y realiza las peticiones a la API de Idealista para obtener todos los inmuebles que cumplen con las características dadas. Una vez descargados, el sistema separa los inmuebles separa según el nivel de búsqueda y por cada zona que se obtiene se calcula sus estadísticas y se descartan las zonas que no cumplan

con los filtros de población.

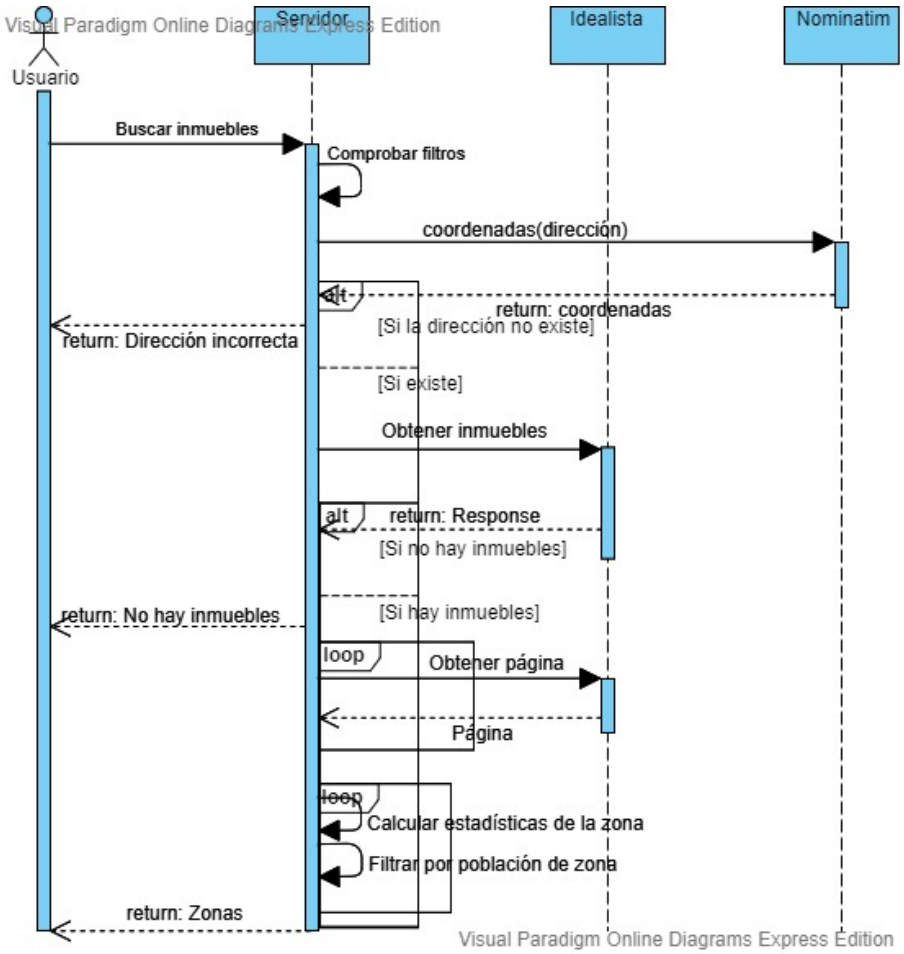

Figura 3.2: Diagrama de secuencia del proceso de calcular el rendimiento de los inmuebles de la búsqueda

3.8. Interfaz

Pantalla principal

La página principal es la pantalla que se muestra al usuario en primer lugar nada más entrar en la aplicación. Desde esta pantalla el usuario tiene la posibilidad de ir a las dos secciones principales de la aplicación, la sección de búsqueda y la sección de recomendación. Existe una parte común a todas las pantallas y es la barra de navegación superior. Desde esta barra de navegación, el usuario tiene la posibilidad de ir a la pantalla principal clicando en el logotipo de la aplicación, de ir a la página web de Idealista clicando en el botón que lleva su nombre, de ir a la pantalla inicial de la sección de búsqueda clicando en el botón Buscar o de ir a la pantalla inicial de la sección de recomendación pinchando en el botón Recomendar. Estos 4 botones se encuentran en la parte izquierda de la barra de navegación y se esconden dentro de un menú hamburguesa si la pantalla es lo suficientemente estrecha como la de

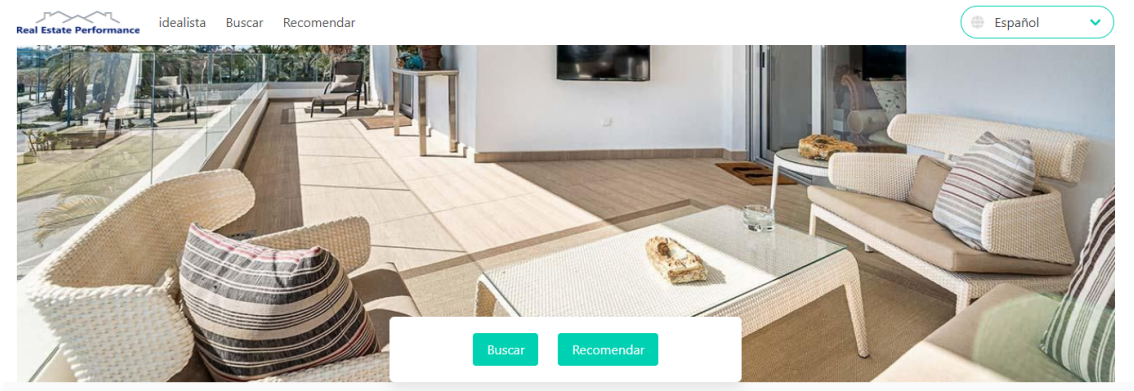

Figura 3.3: Página principal de la aplicación

un móvil. En la parte derecha de la barra de navegación se encuentra un desplegable donde se puede seleccionar el idioma de la aplicación pudiendo escoger entre inglés o español. Esta pantalla se puede ver en la figura 3.3.

Pantalla inicial de la sección de búsqueda

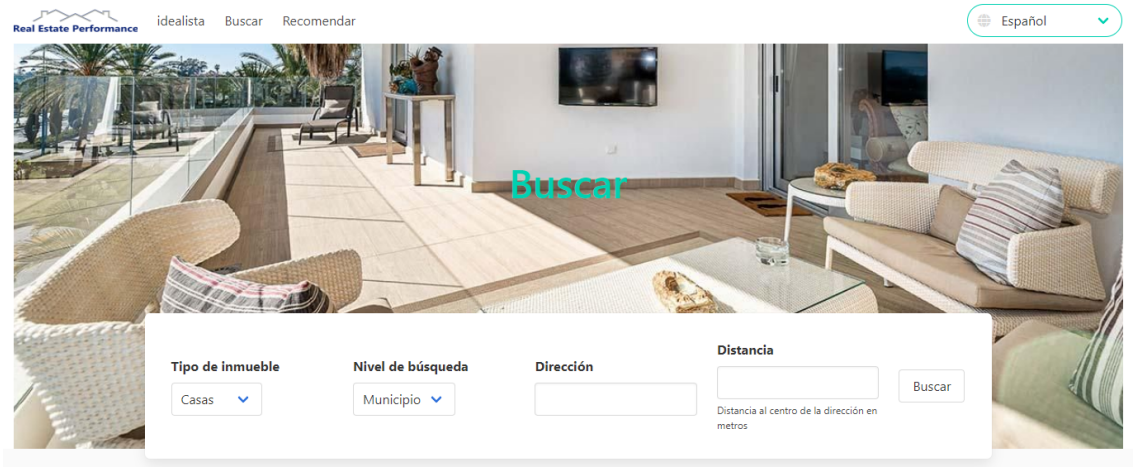

Figura 3.4: Página inicial de la sección de búsqueda de la aplicación

En esta pantalla, el sistema muestra al usuario los 4 filtros básicos necesarios para empezar la búsqueda de inmuebles. Estos filtros empiezan por el filtro del tipo de inmueble que se desea buscar seguido del filtro del tipo de nivel de búsqueda. Por último, se muestran los filtros correspondientes al área de búsqueda que proporcionan al usuario un campo de texto donde escribir la dirección y un campo de texto específico para números donde se introduce el radio o distancia de búsqueda en metros que hay desde el centro de la dirección. Una vez introducidos los datos correctamente y se ha pulsado en el botón Buscar, la aplicación mostrará la página principal de búsqueda con los resultados. La pantalla inicial de la sección de búsqueda se puede ver en la figura 3.4.

Pantalla inicial de la sección de recomendación

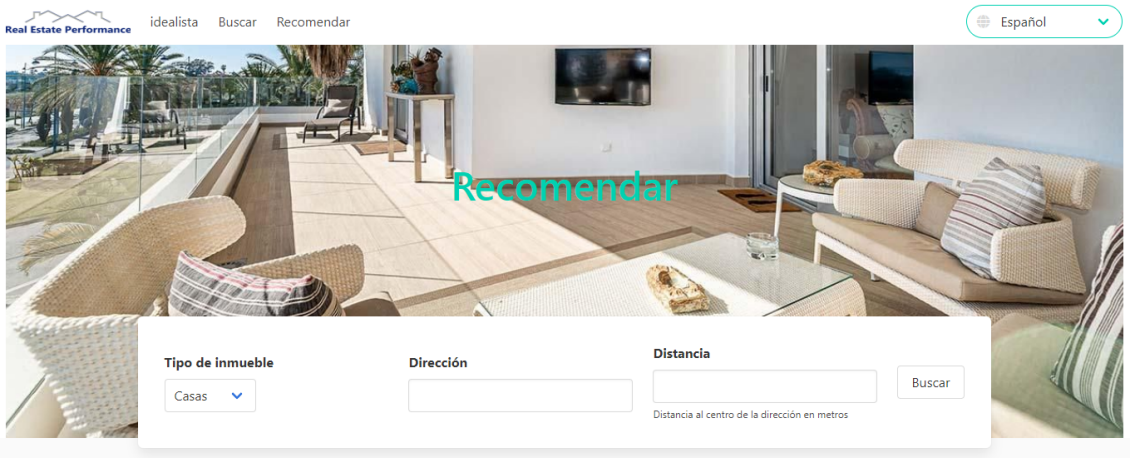

Figura 3.5: Página inicial de la sección de recomendación de la aplicación

Esta pantalla muestra al usuario todos los filtros básicos que se necesitan para empezar una recomendación. La pantalla es muy parecida a la pantalla inicial de búsqueda salvo que el filtro del nivel de búsqueda no está ya que no se necesita y la pantalla mostrará la página principal de recomendación en caso de que los campos sean válidos y el usuario clique sobre el botón Recomendar. Se puede ver esta pantalla en la figura 3.5

Pantalla principal de la sección de búsqueda

La imagen de esta pantalla se puede encontrar en la figura 3.6. Esta pantalla es una de las dos pantallas más importantes de la aplicación. La pantalla muestra los resultados de la evaluación de los inmuebles. Esta pantalla contiene diversas secciones cada una enfocada en un objetivo.

Para empezar, en la parte superior de la pantalla, debajo de la barra de navegación, se puede encontrar una caja donde se informa de la acción que se está realizando que en este caso es la búsqueda (Buscando) y el tipo de inmueble que se ha buscado que en este caso son casas. Debajo se encuentra la zona donde se muestra con palabras la dirección completa que se ha extraído de la dirección que ha proporcionado el usuario.

Debajo de esta caja, se puede ver que se ha dividido la pantalla en dos columnas. En la columna de la izquierda se muestra la caja que contiene todos los filtros que se pueden aplicar a la búsqueda de viviendas. Esta caja varía su contenido dependiendo del tipo de inmueble mostrando siempre los filtros que se pueden aplicar a ese tipo. Para ver todos los filtros, se puede consultar el anexo C.

En la columna de la derecha se encuentra todos los resultados de búsqueda. En primer lugar, se muestra el mapa del área del circulo donde se han buscado todos los inmuebles. El usuario puede desplazarse por el mapa, así como hacer zoom para comprobar si es el área que le interesa buscar. Debajo de este mapa, se puede consultar el resumen de todos los filtros que se han aplicado en la

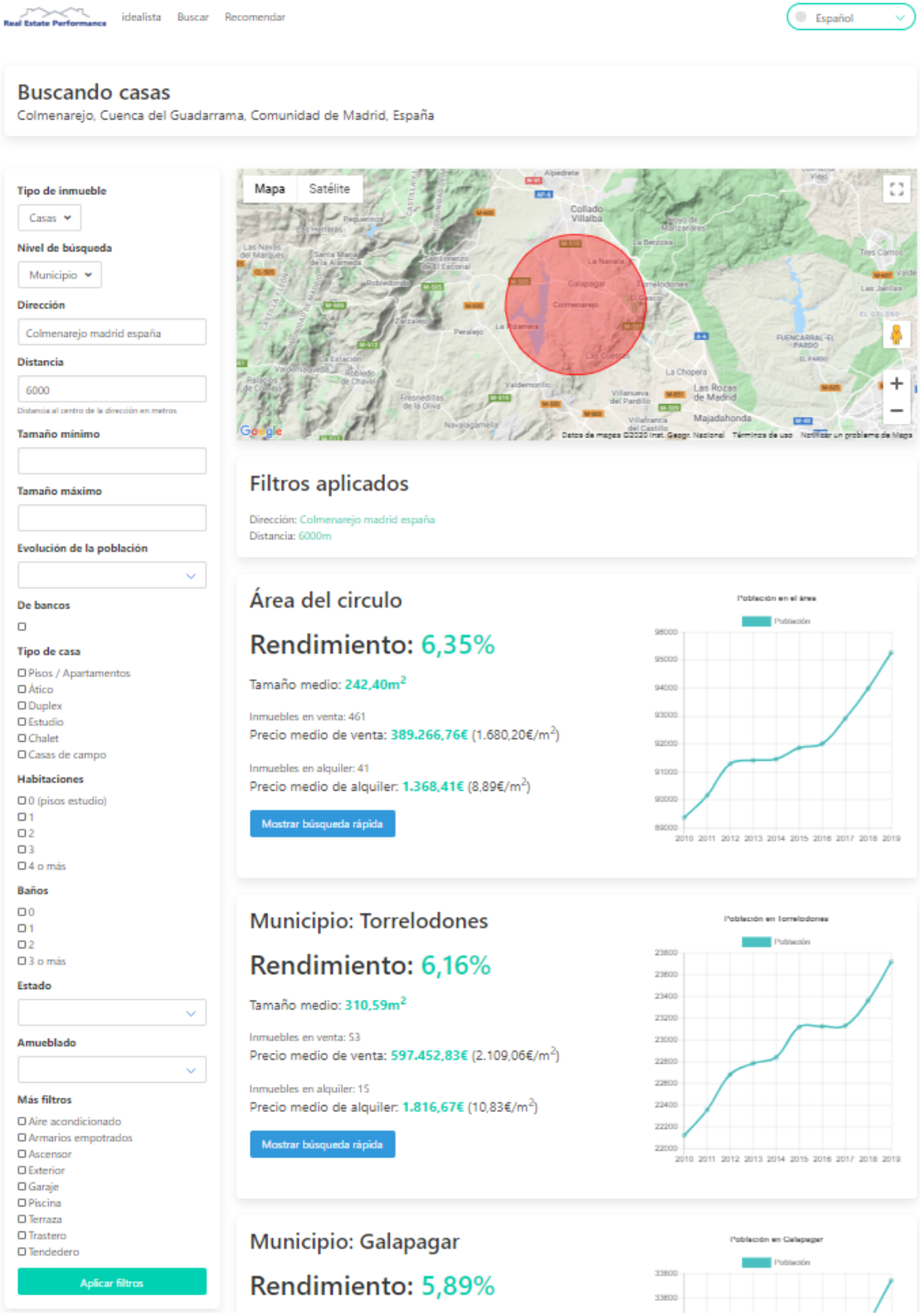

Figura 3.6: Página principal de la sección de búsqueda de la aplicación

búsqueda actual.

Debajo del resumen de los filtros, se encuentra la zona más importante de la pantalla, las estadísticas de los inmuebles de las diferentes zonas. Estas estadísticas estarán dentro de una caja que se corresponde con su zona y estarán ordenadas de mayor a menor rendimiento, pero en primer lugar siempre se encontrará la caja con todas las estadísticas de los inmuebles del área.

Cada caja de estadísticas está compuesta de diversos componentes. La caja está dividida en dos columnas. En la columna de la izquierda se muestra las estadísticas de los inmuebles. Estas estadísticas comienzan por mostrar la zona donde se encuentran mostrando primero el nivel de búsqueda y después el nombre de la zona. Debajo del nombre se puede encontrar la estadística más importante, el rendimiento potencial del alquiler en la zona. Debajo del rendimiento se encuentra la media del tamaño de los inmuebles de la zona, el número de inmuebles en venta y su precio medio total y por metro cuadrado, el número de inmuebles en alquiler y su precio medio en total y por metro cuadrado. Debajo se encuentra un botón que, al interactuar con él, se muestra una tabla comparativa a modo de búsqueda rápida. En la columna de la derecha se muestra el gráfico de la evolución de la población en la zona. En el título del gráfico se muestra el municipio, provincia o área de la evolución de la población.

Pantalla principal de la sección de recomendación

La imagen de esta pantalla se puede encontrar en la figura 3.7. Esta pantalla es la otra de las dos pantallas más importantes de la aplicación y muestra la recomendación de los inmuebles según la búsqueda.

Esta pantalla comparte muchas partes en común con la pantalla principal de la sección de búsqueda. Una diferencia en cuanto a los filtros es que en esta pantalla el filtro del nivel de búsqueda y el filtro de la población desaparecen ya que solo se muestra las recomendaciones de los inmuebles dentro del área y no por cada zona. A cambio se proporciona un filtro para filtrar por precio mínimo y máximo.

Otra diferencia importante se puede ver en la caja de resultados. Esta caja es muy parecida a la caja de resultados en la sección de búsqueda, pero en vez de proporcionar una tabla en la búsqueda rápida, en esta sección se muestra esa misma búsqueda rápida pero los valores en vez de ser el rendimiento ahora son los 3 mejores inmuebles que se puede recomendar y en vez de estar en una tabla están en forma de lista. Por cada inmueble recomendado, se puede ver en la columna de la izquierda una foto de este y en la derecha se puede ver el título, su precio, su tamaño, su rendimiento en alquiler y un enlace al inmueble en el portal inmobiliario donde fue encontrado.

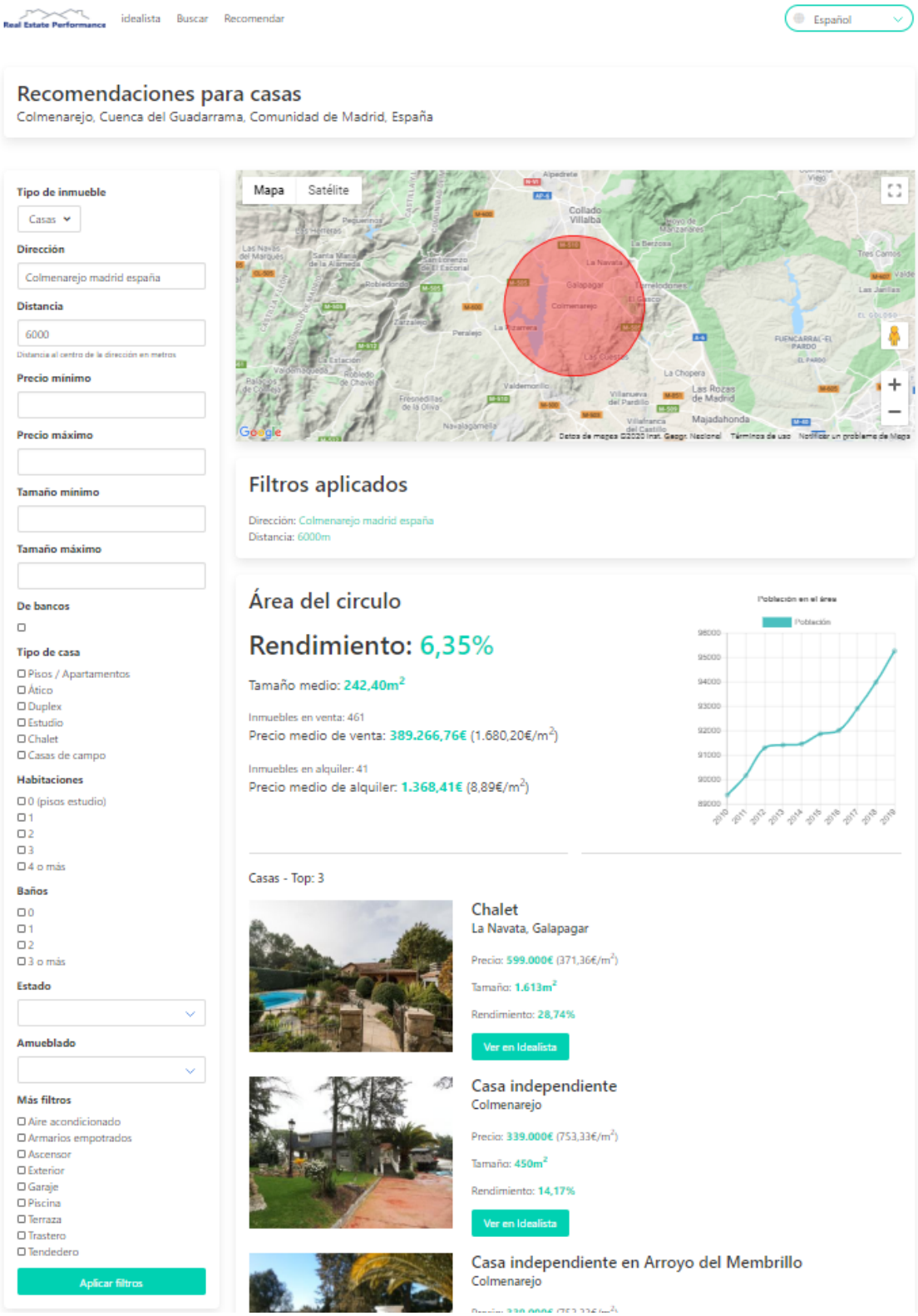

Figura 3.7: Página principal de la sección de recomendación de la aplicación

3.9. Diagrama de Gantt

En la figura 3.8 se puede encontrar el diagrama de Gantt que resume la planificación de todo el desarrollo del proyecto. Esta planificación se ha dividido en los diferentes subsistemas de los que se compone la aplicación con el añadido de la realización de esta memoria. Se ha decidido realizar el desarrollo por diferentes subsistemas para tener un producto usable en la menor cantidad de tiempo posible y para enfocarse en un objetivo específico en cada momento.

En cuanto a la planificación del desarrollo, se ha empezó por realizar pruebas para aprender cómo funcionan las principales tecnologías y servicios y así comprobar su viabilidad. Se empezó probando la API que ofrece Idealista realizando algunas peticiones de prueba para ver cómo funciona, pero sobre todo se prestó especial interés en la obtención del *token*. Una vez que se sabía cómo funciona la API, se empezó a aprender el funcionamiento del INE y su API. Este aprendizaje duró bastante tiempo debido al enorme tamaño del INE y a la deficiente documentación de su API.

Después de aprender cómo funcionaba cada servicio, se empezó a desarrollar el subsistema de búsqueda. Cuando para el desarrollo de este subsistema se necesitaba funcionalidad de otro subsistema que no estuviese terminado del todo, este se terminaba al mismo tiempo. Tras terminar el desarrollo del subsistema de búsqueda, se empezó el del subsistema de recomendación. Para este subsistema, se reutilizó parte de la funcionalidad del subsistema de búsqueda debido a que estos dos subsistemas comparten varias características como la obtención de inmuebles, el mapa o el gráfico de la población. Para este subsistema, solo se tuvo que desarrollar los filtros específicos para este, sus páginas HTML y el cálculo de la recomendación. Por último, solo faltaba desarrollar el subsistema de *scraping* como alternativa a la API de Idealista. Como con los subsistemas de Idealista y de población, primero se empezó haciendo pruebas para aprender cómo funciona el *framework* y después se empezó a implementarlo en Scrapy.

Cuando se terminaron todos los subsistemas y las pruebas de cada uno se habían realizado, se empezó la realización de esta memoria que se desarrolló en tres partes. La primera se corresponde con el desarrollo de la introducción, el estado del arte, las pruebas y resultados y las conclusiones. En la segunda parte se desarrolló el análisis, el diseño y el desarrollo. En la última parte, se hicieron varias iteraciones a cada uno de los apartados para mejorarlos y corregir errores.

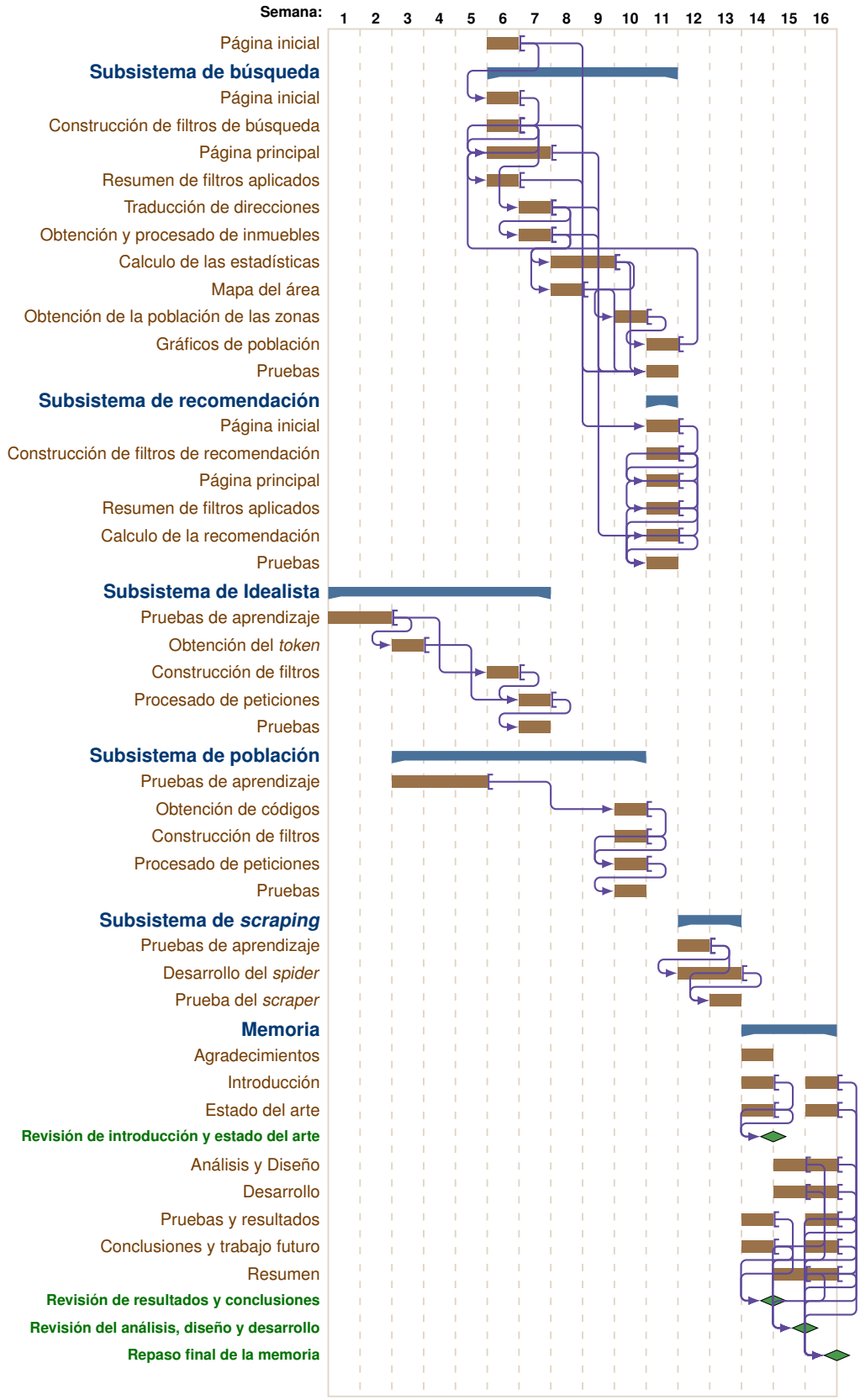

Figura 3.8: Diagrama de Gantt que muestra la planificación del desarrollo del proyecto.

4

Desarrollo

En la fase de desarrollo se van a explicar las diferentes tecnologías que se han usado para implementar y desarrollar cada uno de los subsistemas de la aplicación. Antes de explicar cada una de las herramientas de cada subsistema hay que explicar la elección de la herramienta principal de la aplicación, el *framework* que provee todo lo necesario para ofrecer las páginas web y en este caso, la elección ha sido Django.

Django es un *framework* de desarrollo de páginas y servicios web de código abierto escrito en Python [22]. La elección de este *framework* se debe a diversos motivos. Por una parte, usa Python que es un lenguaje muy potente, común y fácil de aprender y desarrollar [23]. Python es un lenguaje de programación con una comunidad de desarrolladores enorme por lo que encontrar librerías, APIs y *frameworks* que agilizasen el desarrollo sería muy sencillo. Por el contrario, Python es un lenguaje interpretado, por tanto, el rendimiento no es su fuerte y solo sería recomendable si la sencillez y la velocidad son más importantes que el rendimiento. La elección de Django también se debe a su sencillez y facilidad de su arquitectura. Otra ventaja muy importante es que permite desarrollar un producto muy rápido y este llegará a muchas más personas que si por ejemplo se desarrollase una aplicación móvil. Las desventajas de una página web son la poca integración con los diferentes sistemas operativos que existen y el rendimiento. Python y Django, aparte de ofrecer las ventajas comentadas, son el lenguaje y el *framework* con el que el equipo de desarrollo está más familiarizado.

Una de las ventajas o inconvenientes de Django, según se mire, es ORM que es una técnica de programación que convierte los objetos propios de un lenguaje de programación orientado a objetos a tipos compatibles con una base de datos. Si lo resumimos, una clase en Django se corresponde con una tabla en la base de datos y un objeto con una fila y convertirá las listas de objetos a referencias de otra tabla a la tabla de la clase. Django posee una técnica para buscar objetos en una base de datos sin usar el lenguaje de SQL (Structured Query Language o lenguaje de consulta estructurada en español). Django provee de métodos de clase que permiten filtrar objetos mediante filtros como los de igualdad y ordenar, agrupar y limitar el resultado.

Otra de las principales ventajas que ofrece Django son los formularios. Django permite crear formularios de una manera muy sencilla semejante a la creación de modelos. Los formularios se construyen

indicando el nombre de la variable y el tipo de entrada que se desea incluir (texto, opciones, fechas, etc.). Se puede especificar el tipo de valor inicial como también el *widget* que se mostrará. En el anexo D se puede ver un ejemplo de la declaración de un formulario.

Para que Django funcione correctamente, se necesita de una base de datos que sea compatible con él. Por defecto, Django viene equipado con la base de datos SQLite. Este gestor de base de datos se caracteriza por ser ligero y fácilmente configurable debido a que se basa en ficheros locales que se guardan junto a la aplicación. Aunque SQLite es una buena base de datos para proyectos pequeños, no es acta para grandes proyectos y tampoco ofrece una gran variedad de tipos. Por estas razones, se ha decidido usar el gestor de bases de datos PostgreSQL.

PostgreSQL es un gestor de base de datos relacional de código abierto con una gran comunidad [24]. Una de sus principales características es que funciona en los principales sistemas operativos de escritorio a la vez que proporciona un gran rendimiento y una alta concurrencia con un consumo de recursos bastante bajo. Otra característica muy importante son sus tipos extra que ofrece. PostgreSQL proporciona más tipos de los proporcionados comúnmente por los gestores de bases de datos como el campo JSON que permite guardar en él documentos JSON y permite buscar variables dentro de este campo como si se tratase de una columna más de la tabla.

Su elección recae en el conocimiento que tiene el equipo de desarrollo en él. Es un software que el equipo conoce muy bien lo que ha permitido rapidez a la hora de integrarlo. También se ha elegido debido a su bajo consumo de recursos para su gran potencial.

Como administrador de PostgreSQL se ha elegido a PgAdmin [25]. Este administrador ha sido especialmente diseñado para trabajar con este gestor de base de datos y esta ha sido la causa principal de su elección ya que ofrece características que están enfocadas completamente en él.

Una vez explicado que es Django, el componente principal de la aplicación, y el porqué de su elección dada sus ventajas y desventajas toca exponer las herramientas y servicios que se han usado e implementado en cada uno de los subsistemas.

4.1. Subsistema de búsqueda y subsistema recomendación

Estos dos subsistemas comparten gran parte de su código excepto por pequeños añadidos que uno tiene y el otro no. Para ambos módulos se han usado herramientas comunes como la generación de gráficos de población, la vista del mapa o la apariencia de la página.

Para desarrollar las páginas web y que estas tuviesen un estilo moderno, simple y elegante se ha decido usar un *framework* CSS que simplifica en gran medida el desarrollo a la vez que reduce su tiem-

po. Una gran ventaja de usar *frameworks* CSS para dar estilo a las páginas web es su compatibilidad con todos los navegadores populares debido a que solo se centran en el estilo. Otra ventaja importante es su popularidad. La mayor parte de las páginas web utilizan estos *frameworks* para dar estilo a sus páginas y causando que los usuarios se familiaricen con él, pero el estilo también es una desventaja ya que casi todas las páginas web lucen igual y no tengan personalidad propia. Otra desventaja, que se va reduciendo a medida que las conexiones a internet son más rápidas, es el tamaño. A mayor tamaño más tiempo tarda en enviarse al cliente. Por lo general, los *frameworks* CSS ocupan más espacio que si se usase una hoja de estilo CSS desarrollada especialmente para la aplicación, pero llevaría mucho tiempo.

Como *framework* CSS se ha elegido a Bulma [26]. Bulma es un *framework* bastante moderno y recientemente. Está basado completamente en tecnología *flexbox*, nueva en la versión 3 de CSS. Bulma es de código abierto y se puede personalizar mediante variables. A diferencia de otros *frameworks* CSS, Bulma solo proporciona ficheros CSS y no proporciona ningún tipo de *script* causando que esté muy optimizado y que pueda mejorar con mucha rapidez. La principal desventaja de esto es que, para algunas opciones como los modales, es necesario escribir uno mismo los *scripts* para abrirlo y cerrarlo. La principal decisión para escoger Bulma es su sencillez y modernidad comparados con otros *frameworks* CSS.

Para el desarrollo de estos subsistemas, hace falta traducir la dirección proporcionada por el usuario a coordenadas geográficas que entienda el subsistema de Idealista. Para hacer esta traducción hay que usar un servicio de terceros y tras analizar todas las alternativas se ha decidido usar la librería Geopy y el servicio de mapas OpenStreetMaps.

Geopy es una librería escrita en Python que hace de intermediario entre un cliente y diversos sistemas de localización basados en direcciones o coordenadas [27]. Geopy no proporciona directamente los servicios, sino que posee una gran variedad de ellos y los unifica bajo una misma librería. Esto simplifica mucho el trabajo cuando un servicio de localización de direcciones no funciona correctamente o no es del agrado de los desarrolladores. Como servicio proveedor de traducciones se ha usado Nominatim [28]. Nominatim es un proveedor de traducciones de direcciones basado en datos de OpenStreetMaps. La ventaja de este servicio es que es abierto y provee de servicios de manera gratuita. La elección de Geopy recae en sencillez que ayuda a que el desarrollo sea rápido y su gran lista de proveedores que ayuda a elegir el mejor de ellos y cambiar fácilmente entre ellos.

Para estos subsistemas también hace falta una librería que muestre diferentes gráficos para enseñar la evolución de la población. Tras estudiar y analizar las principales herramientas, se ha decidido usar la librería Charts.js.

Charts.js es una librería escrita en JavaScript que permite mostrar gráficos de diferentes estilos [29]. Esta librería se distribuyen bajo código abierto y permite representar gráficos en cualquier navegador bajo canvas. Charts.js es muy sencilla de utilizar y bastante personalizable. Una de sus principales

ventajas es su ligereza y velocidad, pero no descuida otras aspectos como la documentación. La variedad es una de sus desventajas al no disponer de muchos estilos y tipos de gráficos. Se ha elegido esta librería por su ligereza y sencillez ya que la poca variedad que posee no es un problema debido a la sencillez de los gráficos que se han de mostrar.

Por último, para estos dos subsistemas hace falta una herramienta que muestre el mapa del área. Este mapa se usará para mostrar al usuario la ubicación de la búsqueda y su radio. Se ha decidido usar Google Maps como proveedor de mapas ya que es el más popular en estos tiempos haciendo que la mayoría de los usuarios se sientan cómodos gracias a su familiaridad y conocimiento.

Google Maps es una aplicación que provee de diversos servicios desde la búsqueda de un lugar de interés hasta la función de navegador. Uno de los servicios que más interesan para la aplicación es la API que proporciona la vista en un mapa de un lugar o lugares especificados [30] de diferentes formas. El servicio funciona mediante *script* en el navegador del cliente y es este quien lo muestra evitando hacerlo desde el servidor. Este servicio no es gratuito y cada petición que se hace para cuesta dinero, pero ofrecen como regalo de bienvenida un saldo de 300$ que caduca al año.

4.2. Subsistema de Idealista

Este subsistema es el encargado de obtener los inmuebles utilizando diferentes filtros. Los inmuebles se pueden obtener de dos lugares diferentes pero procedentes de la misma base de datos. El primer lugar del que se pueden obtener es mediante la API de Idealista y el segundo lugar es la base de datos local que se ha creado mediante el *crawling* de la página web de Idealista. La mayor parte de la funcionalidad de este sistema consiste en la obtención de inmuebles mediante peticiones a la API.

La API o interfaz de programación de aplicaciones de Idealista [31] es un servicio web que ofrece el portal inmobiliario y se define como “un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones" [32]. Para tener acceso a la API lo primero que hay que hacer es ir a su página web y enviar una solicitud indicando nombre, correo y motivo. Una vez procesada y concedida, envían un correo con la documentación para la obtención del *token* Bearer mediante el usuario y la contraseña proporcionados y la documentación que describe los servicios que ofrece y el formato tanto de la petición como de la respuesta.

De manera gratuita, Idealista ofrece 100 peticiones al mes y 1 petición por segundo. Esto es una gran limitación para nuestro sistema ya que para que funcione correctamente ha de buscar todos los inmuebles disponibles en el área y dependiendo de esta el número de inmuebles pueden superar el número máximo de peticiones. Para arreglar está situación, se contactó con Idealista pidiendo más peticiones gratuitas pero debido a la situación excepcional que se ha vivido estos últimos meses no se obtuvo respuesta alguna a la petición por lo que se buscó otra forma de obtener los inmuebles mediante el *scraping* de su página web.

La API de Idealista utiliza los servicios web para la comunicación con el protocolo REST (Representational State Transfer- Transferencia de Estado Representacional) que se basa en HTTP y URL para pedir información. Para obtener los datos de Idealista no hay más que escribir la dirección URL de su servicio que es `https://api.idealista.com/3.5/es/search` especificando los filtros de búsqueda, que se definen como parámetros GET (Ej. `https://api.idealista.com/3/es/search?locale=es&maxItems=20&numPage=1&operation=sale&order=publicationDate&propertyType=garages&sort=desc&apikey={api_key}&t=1&bankOffer=true`).

En cada una de las peticiones hace falta indicar el *token* Bearer para la identificación y para su obtención hace falta enviar una petición POST HTTP a la URL `https://api.idealista.com/oauth/token` en la que se declare `Content-Type: application/x-www-form-urlencoded` que indica el tipo de petición que se hace y `Authorization: Basic YWJjOjEyMw==` que indica el usuario y contraseña codificados según el estándar RFC 1738 [33]. Este estándar especifica que el usuario y la contraseña han de ir codificados en Base64 de tal forma que el usuario y la contraseña estén de esta forma `base64("usuario:contraseña")`. El servidor devolverá en caso de éxito un JSON en la que se especifica la clave y el tiempo de expiración en segundos.

Una de las dudas que surgieron en el desarrollo de este subsistema fue el método para enviar las peticiones. Tras investigar todas las posibilidades que existen se eligió usar la librería requests que ofrece Python.

Requests es una librería escrita en Python que permite enviar peticiones HTTP de manera muy rápida y sencilla solamente indicando la URL correcta para obtener los datos despreocupándose de administrar correctamente las cabeceras y las respuestas [34]. Al integrar esta librería con el sistema y con la API de Idealista surgió una necesidad no planteada con anterioridad. El problema que surgió consistía con la correcta utilización y administración del *token* de identificación que hay que enviar con cada petición. Este *token* hay que actualizarlo si este caduca y la administración de esta fecha de caducidad sería un trabajo extra no planeado. Para solucionar este problema si investigaron librerías que permitiesen la administración automática del *token* y se llegó a la decisión de utilizar la librería requests-outhlib.

Esta sencillez no es tan grande si hablamos de la API de Idealista. Esta API tiene la particularidad de necesitar identificación en cada una de las peticiones. Está identificación se ha de pedir a otro servicio de Idealista y no dura para siempre, sino que caduca pasado un tiempo. Para facilitar el uso de la identificación y su renovación se ha usado otra librería que facilita este trabajo.

Requests-outhlib es una librería creada especialmente para proveer el soporte outhlib a la librería requests [35] simplificando mucho las peticiones a servicios que requieren identificación mediante el uso de usuario y contraseña con el estándar RFC1738. Requests-outhlib proporciona métodos para obtener automáticamente la identificación en caso de que esta haya caducado o no se haya obtenido aún.

En general, trabajar con esta API ha resultado ser sencillo y según lo esperado en el diseño. Hubo algunas complicaciones a la hora de trabajar con las peticiones ya que la documentación no especificaba correctamente el formato de los campos multivalor. En algunos de estos campos se puede especificar varios valores, pero en otros solo se puede especificar un valor y esto no está bien indicado. También hubo problemas con el formato de respuesta. Algunos de los valores se reciben en un formato diferente al especificado y otros valores que se recibían no venían en la documentación.

4.3. Subsistema de población

El subsistema de población es el encargado de todo lo relacionado a la obtención de la población de los municipios y provincias. En la fase de análisis se decidió obtener estos datos del INE o Instituto Nacional de Estadística. Para este proyecto, de todas las estadísticas que proporcionan, se necesita la estadística del censo que se ofrece en dos tablas diferentes desde el año 1996. En una tabla, se muestra el censo por provincias y en la otra el censo por municipios. Ambas permiten filtrar la población por sexo y grupo de edad, pero lo necesario para el proyecto son las cifras totales.

Para obtener estos datos, se ha usado la API REST que proporciona el INE. Para pedir los datos de la población, se necesita obtener primero todos las provincias y municipios de España con su código identificativo ya que la API solo entiende estos identificadores. A diferencia de la API de Idealista, esta no necesita de autenticación y para realizar las peticiones se ha usado la librería requests de Python.

El uso e integración de esta API en el sistema fue más complicado de la inicialmente previsto. El INE posee dos tipos de sistema diferentes de API y no especifican claramente que sistema se necesita para que estadística. Una vez se ha encontrado el sistema correcto, el INE facilita una aplicación que mediante un formulario permite crear la URL que se necesita para obtener los datos. Esta aplicación ha simplificado el desarrollo, pero tiene complicaciones como la falta de documentación o que no crea la URL correctamente sino solo muestra la plantilla correcta y a partir de ahí hay que ingeniárselas para saber que es cada parámetro y para qué sirve. Al final se ha conseguido obtener todas los códigos de provincias y municipios, que se obtienen al iniciar la aplicación y se actualizan cada semana, y se ha podido obtener correctamente los datos necesarios de la población y en el formato adecuado.

4.4. Subsistema de *scraping*

El subsistema de *scraping* es el subsistema que se encarga de obtener todos los datos de inmuebles de la página web de Idealista mediante el método conocido como web *scraping*. Un *scraper* es un programa que se encarga de obtener las páginas web mediante el protocolo HTTP y extraer información del documento HTML. En este proyecto, el *scraper* se usará para extraer todos los datos de un inmueble como su precio, su tamaño, su ubicación, sus características físicas dependiendo del tipo de

propiedad, etc., respetando en la medida de los posible las normas que Idealista ha declarado para los *crawlers* en su fichero robots.txt. El *scraper* se utiliza como alternativa a la API de Idealista que tiene algunas limitaciones en su modalidad gratuita como el límite de peticiones por segundo y el número máximo de peticiones mensuales. Se ha decidido que el *scraper* se implementaría en Python para que esté integrado con el resto de la aplicación, aunque se podría utilizar otro lenguaje. Tras estudiar todos los *scrapers* que existen para Python en la actualizad se ha elegido usar Scrapy.

Scrapy es un *framework* que permite obtener páginas web y extraer información estructura de ellas [36]. Se puede usar para diferentes objetivos desde el control del precio de un producto en un comercio electrónico hasta pruebas con un servicio web. Entre sus ventajas está la facilidad de uso, su gran escalabilidad y su gran diseño ya que proporciona métodos que permiten agregar funcionalidad extra mediante extensiones.

Scrapy puede funcionar de varias maneras y la forma más fácil de manejarlo es mediante su programa de línea de comandos que proporciona varios modos de uso cada uno con una finalidad. El modo *shell* se usa para descargar una página HTML de una dirección y probar a extraer información de ella de modo muy parecido a depurar un programa. A parte del modo *shell*, Scrapy permite crear aplicaciones propias en Python. La parte principal de estas aplicaciones son los *spiders* que se encargan proporcionar las URLs base con las que empezar la extracción de datos y de parsear las páginas que se han descargado y definiendo que método realiza el parseo de que petición. Scrapy permite ejecutar estas aplicaciones de dos maneras. Una es mediante su programa de línea de comandos mediante la función `runspider <spider.py>` y la otra es mediante un *script* Python llamando a funciones específicas de Scrapy para tal efecto.

Para extraer información de la página, Scrapy proporciona dos formas. Una es mediante clases CSS que permite seleccionar los elementos de una página HTML como si se usara desde un fichero CSS. El otro método es XPATH que es un lenguaje que se creó explícitamente para direccionar elementos de documentos XML [37] y funciona de forma diferente a CSS, pero es mucho más potente y versátil.

La base sobre la que se ha desarrollado este *scraper* pertenece al proyecto Dedomeno del usuario Ginopalazzo [38] sobre la cual se han realizado diversas modificaciones para corregir los errores que tenía y para adaptarla a nuestro sistema.

El *scraper* se usa para obtener todos los datos de la página web de Idealista dada una provincia, un tipo de inmueble y una operación. Se empieza buscando la lista con todos los inmuebles de ese tipo y con esa operación en la provincia dada. Una vez se tiene la página, se extrae la lista de todos los enlaces a los diferentes inmuebles y para continuar extrayendo datos, se busca si existe una página siguiente para extraer los datos de esta de manera recursiva. Los inmuebles se extraen mediante otro método que busca las características comunes como la dirección y el precio y mediante otro método para obtener los datos característicos de cada tipo de inmueble.

Para añadir más funcionalidad al *framework*, Scrapy permite añadir *middlewares* que se ejecutan

antes o después de cada petición y se encargan de realizar una amplia variedad de objetivos desde cambiar alguna cabecera hasta comprobar que la respuesta es correcta antes del parseo. Gracias a esta forma de añadir funcionalidad, existen numerosas extensiones que simplifican mucho el desarrollo.

Una de las extensiones que existen para Scrapy que ha simplificado mucho el desarrollo es scrapy-djangoitems [39]. Esta extensión permite declarar Items en Scrapy, que son objetos para guardar datos que se extraen de páginas, relacionándolos con modelos de Django y permite guardar los Items directamente en la base de datos mediante los mismos métodos de Django. Gracias a esta extensión no ha hecho falta declarar los valores necesarios en cada Item, sino que los extrae de los modelos ahorrando mucho tiempo.

Para conseguir que Scrapy pudiera obtener todos los inmuebles correctamente se ha configurado para que realizase una petición cada 5 segundos y una petición simultáneamente. Con estos ajustes, se consiguió descargar con éxito un máximo de 30 inmuebles de Idealista, pero con el tiempo, cada petición se rechazaba con el error HTTP 403 y un mensaje que decía que se estaban haciendo demasiadas peticiones. Se ha probado a aumentar el tiempo entre peticiones, pero al final también eran rechazadas. Esto puede indicar que Idealista está utilizando métodos para detectar *crawlers*. Para evitar de otra forma el rechazo de las peticiones, se han utilizado dos extensiones muy útiles. Una de ellas es scrapy-user-agents que permite definir un agente de usuario diferente para cada una de las peticiones [40]. Cambiar el agente de usuario hace pensar al servidor de Idealista que se trata de otro cliente quien realiza las peticiones, aunque la dirección de la que provengan sea la misma. La otra extensión que se ha utilizado es scrapy-proxies [41]. Esta extensión permite usar una lista de *proxies* para pedir la información desde otro ordenador (proxy) que actúa de intermediario. Los *proxies* funcionan de tal forma que les envías una petición y ellos la reenvían a la dirección que les proporcionas de tal forma que el servidor destino piensa que la petición la ha realizado el proxy y no el cliente que está detrás. El éxito de esta manera de evitar el rechazo de la petición está en usar *proxies* de calidad que la mayoría son de pago y solamente algunos de ellos ofrecen pruebas temporales gratuitas.

5 Pruebas y resultados

5.1. Pruebas

5.1.1. *Setup* inicial de pruebas

Para inicializar el *setup* inicial de pruebas se necesita disponer de un sistema que sea compatible con un intérprete de Python, disponible para descargar gratuitamente en su página web [23], y con el gestor de base de datos PostgreSQL, también disponible en su página web de forma gratuita [24]. Para el desarrollo y las pruebas se ha usado la versión 3.7 de Python y la versión 12 de PostgreSQL. Una vez descargados, hace falta instalar, para evitar incompatibilidades, un entorno virtual de Python. En este caso se ha usado virtualenv descargado mediante el gestor de paquetes de Python llamado PiP. Una vez configurado hay instalar todos los paquetes y dependencias necesarias para la aplicación mediante el gestor de paquetes PiP en el entorno virtual. Junto con el código de la aplicación se provee de un fichero llamado requirements.txt donde se encuentran todos los nombres de los paquetes y sus versiones. Para instalarlos solo hace falta ejecutar el comando `pip install -r requirements.txt`.

Una vez instalado todos los paquetes y la aplicación, hace falta configurar la base de datos. Para ello hace falta crear un usuario con nombre y contraseña alumnodb. Este usuario debe ser el propietario de una base de datos llamada Idealista. Se puede utilizar otro usuario y otra base de datos si se cambia estos parámetros en el fichero settings.py de Django. Para crear todas las tablas desde Django, hace falta ejecutar el comando `python manage.py makemigrations` que analiza todos los modelos de la aplicación y el comando `python manage.py migrate` que crea las tablas en la base de datos.

En el git también están disponible dos ficheros JSON que guardan todos los inmuebles descargados mediante la API de Idealista haciendo una búsqueda de viviendas en el municipio de Colmenarejo con un radio de 6.000 metros. La finalidad de estos ficheros es la de evitar hacer demasiadas peticiones a la API de Idealista a la hora de hacer las pruebas para evitar el gasto innecesario de peticiones gratuitas. Se guardan en este formato para simular lo máximo posible el formato de datos que devuelve la API.

Para probar la página web no hay más que ejecutar el servidor de prueba de Django mediante el

comando `python manage.py runserver`. Una vez iniciado, en la terminal se mostrará el enlace para acceder a la página principal. Desde ahí se puede probar la aplicación y realizar diferentes pruebas simulando la acción real. Las búsquedas de inmuebles siempre devolverán el mismo resultado ya que se están usando los ficheros de pruebas por defecto. Para buscar los inmuebles mediante la API de Idealista hace falta descomentar su código y seleccionar el usuario y contraseña válidos y comentar la sección de carga de ficheros. Al proporcionar pocas peticiones gratuitas, no se recomienda hacer muchas pruebas y grandes búsquedas.

Para ejecutar el *scraper* solo hace falta ejecutar desde el directorio del proyecto el comando `scrapy runspider idealista_scrapy/spiders/province_properties_spyder.py` y empezará a descargar los datos de los inmuebles que se hayan especificado. Para cambiar la búsqueda solo hace falta cambiar los argumentos por defecto que se proporcionan en el fichero province_properties_spyder.py.

En cuanto al hardware utilizado para realizar las pruebas, se ha usado un ordenador de sobremesa con un Ryzen 5 3600 de 6 núcleos y 12 hilos a una frecuencia base de 3,6GHz y una turbo de 4,2GHz, 16GB de memoria RAM DDR4 a 3200MHz y una conexión a internet de 600Mb simétricos. En cuanto al sistema operativo, se han realizado las pruebas en Windows 10 versión 2004 y en Manjaro Linux sobre kernel Linux versión 5.7. Ambos sistemas tienen las mismas versiones y paquetes que se han especificado.

5.1.2. Pruebas de verificación

Pruebas de Caja Negra

En este tipo de pruebas solo se tiene en cuenta la entrada y la salida del sistema sin tener en cuenta la lógica que hay detrás. Para probar toda la funcionalidad, se ha procedido a realizar las pruebas de búsqueda en la que se ha buscado siempre en la misma dirección y con la misma distancia. Solamente se ha variado el tipo de inmueble y el nivel de búsqueda y solo se ha probado una vez a cambiar de dirección para comprobar la extracción de las coordenadas.

Para comprobar que el sistema calcula correctamente todas las estadísticas, se ha usado una base de datos de prueba cuyas estadísticas se conocían de antemano. Esta base de datos son los dos ficheros JSON. Para probar los inmuebles que no son viviendas se ha usado directamente la API de Idealista ya que no se han guardado los datos devueltos. Con esta base de datos, se ha podido verificar que el sistema calcula correctamente las estadísticas de todos los niveles de búsqueda y, por usar el mismo código, todos los tipos de inmuebles. Para verificar el correcto funcionamiento de las recomendaciones, a la base de datos de prueba se le ha añadido un inmueble que destaca de los demás por tener un precio muy inferior por lo que ese inmueble tendría que recomendarse en primer lugar.

Todas las pruebas de caja negra se han verificado correctamente y funcionan según lo esperado.

Pruebas de Caja Blanca

Estas pruebas consisten en analizar y verificar la lógica de la aplicación. En estas pruebas se intenta que al menos todas las líneas de código se ejecutan al menos una vez. Como son pruebas muy pesadas y que conllevan mucho tiempo, se suelen realizar en módulos pequeños.

Para el análisis, se ha empleado la página web para introducir los datos necesarios para probar cada línea de código. Para probar el control de los errores, estos se han simulado mediante código que directamente devuelve el error. En esta prueba no se ha usado la base de datos de prueba, sino que se han realizado las peticiones directamente a Idealista para probar este sistema. También se han probado que los sistemas de obtención datos de la web mediante *scraping* funcionan correctamente. Para ello, se ha usado un *script* de prueba que ejecuta el *crawler* y mediante depuración se ha probado toda la lógica.

Todas las pruebas de caja blanca se han verificado correctamente y funcionan según lo esperado.

5.1.3. Pruebas de validación

Las pruebas de validación consisten en comprobar que todos los requisitos especificados se cumplen. Para comprobarlo, se han repasado uno por uno los requisitos funcionales y comprobado que dicha funcionalidad existe y funciona. Para los requisitos no funcionales se han comprobado aquellos requisitos que se pueden medir como la compatibilidad con los navegadores más populares probando las páginas web en cada uno de ellos excepto Safari. Todos los requisitos funcionales y no funcional han sido comprobados y se han implementado.

5.2. Resultados

Para obtener los resultados, se ha probado a realizar una búsqueda sencilla en la página web. En la sección de búsqueda, se ha filtrado los inmuebles por viviendas con un nivel de búsqueda por municipios. En cuanto a la ubicación, se ha usado como dirección el municipio de Colmenarejo poniendo en la dirección “Colmenarejo madrid españa". Como distancia de búsqueda se ha usado 6000 metros. Como resultado, se ha obtenido un resumen con todas las métricas de los inmuebles y su rentabilidad en el área de búsqueda, así como 4 municipios que tienen viviendas en venta y alquiler dentro del área del circulo. Estos 4 municipios son: Colmenarejo, Galapagar, El Escorial y Torrelodones. En cuanto al rendimiento, el municipio con mayor rentabilidad potencial es el de Torrelodones con un 6,16 %.

En cuanto a los resultados de recomendación, se han usado los filtros anteriores y se ha obtenido

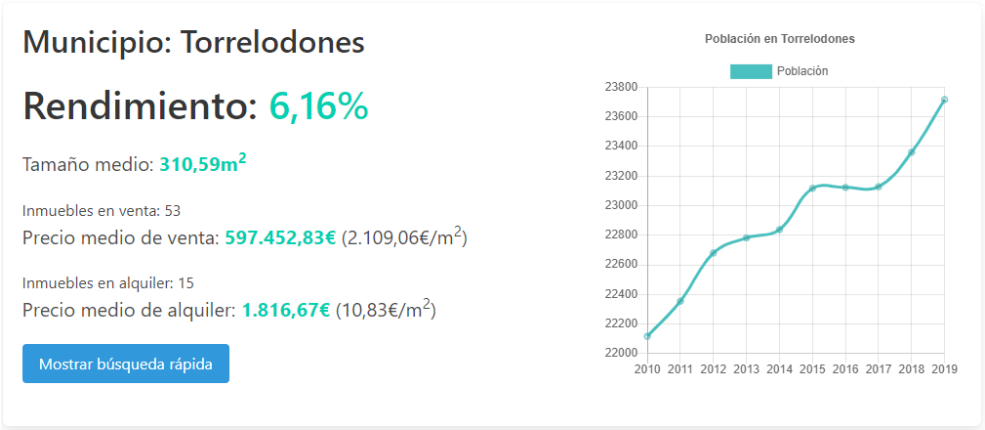

Figura 5.1: Captura de pantalla de la aplicación con las estadísticas del municipio de Torrelodones

los mejores inmuebles para invertir. En la figura 5.2 se puede ver que el mejor inmueble para invertir es un chalé situado en La Navata, Galapagar, que ofrecería un rendimiento potencial del 28,74 % debido a su gran tamaño y a su bajo precio de venta.

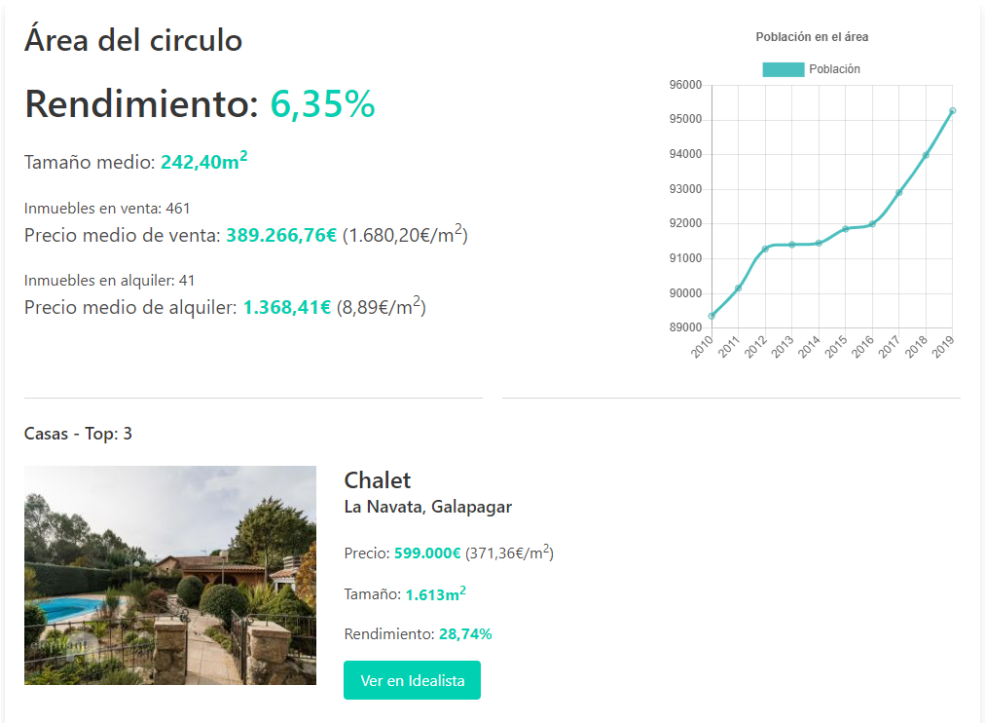

Figura 5.2: Captura de pantalla de la aplicación con las recomendaciones de inmuebles en el área del circulo

6

Conclusiones y trabajo futuro

6.1. Conclusiones

En conclusión, este trabajo de fin de grado ha sido muy interesante porque se ha podido trabajar con tecnologías web muy variadas y con mucho potencial muy usadas hoy en día por multitud de empresas e instituciones.

Consideramos que este trabajo ha sido un éxito porque se ha conseguido los objetivos fijados, es decir, se dispone de una aplicación web fácil y sencilla a través de la cual se pueden buscar inmuebles o propiedades por zonas y diferentes criterios basados en sus características y observar y evaluar las diferentes rentabilidades potenciales que hay en cada zona.

Para conseguir que la aplicación sea rápida, sencilla y fácil de desarrollar y de usar se han aprovechado diferentes *frameworks* como Django que es una tecnología moderna, muy usable, sencilla, fácil y rápida de desarrollar que ha permitido desarrollar, depurar, testear e iterar de una forma rápida y fácil. El desarrollo de la página con Django ha sido todo un acierto ya que todo lo que promete se ha cumplido en este proyecto.

Por otro lado, se ha conseguido aprovechar fuentes de información online como el INE o Idealista. El INE es un gran recurso a disposición de cualquiera que está disponible online de forma gratuita y que provee de una API para facilitar el acceso desde aplicaciones de terceros. La base de datos del INE es enorme y su adaptación a la aplicación ha sido más difícil de lo inicialmente planeado debido a su complejidad, su mayor desconocimiento de lo previsto y sobre todo a su falta de documentación causada por tener dos sistemas diferentes de API y a las diversas formas, todas ellas complejas, de acceder a un mismo recurso.

Otra información online que se ha utilizado es Idealista que es una fuente privada que se ofrece de forma pública y abierta mediante su portal inmobiliario que es número uno en España. Idealista proporciona una API para que los desarrolladores puedan utilizarla para realizar búsquedas de inmuebles de forma sencilla. Esta API está protegida mediante usuario y contraseña y ofrece solo 100 peticiones de forma gratuita. Su utilización fue bastante sencilla y según lo planificado ya que su documentación está

bien documentada en cuanto a las características que ofrece y el método de obtención de la autenticación pero ocurrieron problemas no planeados debido a que algunos parámetros de las peticiones se enviaban y recibían de forma diferente a lo documentado.

Por las limitaciones que tiene la API Idealista en su modalidad gratuita, se implementó un *scraper* de su página web como alternativa. Este desarrollo no estaba planificado en primer lugar y su desarrollo fue sencillo en algunos elementos y complicados en otros. La documentación de Scrapy facilitó mucho el trabajo de diseño y pruebas para crear una base de datos propia, pero a la hora de obtener todos los datos de los inmuebles resultó muy complicado debido a la protección *antiscrapers* que implementa Idealista y que tuvo que solucionarse a base de cambiar los identificadores de los agentes de usuario en las peticiones y a base de usar proxies gratuitos. Aunque el uso de Idealista tiene limitaciones o es complicado de usar, su gran base de datos de inmuebles merece el esfuerzo.

Si no se hubieran utilizado estas herramientas y servicios, no se hubiera podido tener un producto de calidad, simple y sencillo de usar de manera rápida y sencilla.

6.2. Trabajo futuro

Esta aplicación es una aplicación web y dado que usa este tipo de tecnologías, es muy probable que llegue a la mayor cantidad de gente que otras tecnologías, pero en un futuro se podría implementar una aplicación para los principales sistemas operativos móviles de forma que los usuarios tuvieran una mejor opción a la hora de usar sus dispositivos. Implementar una aplicación móvil a parte de ofrecer el servicio web haría que las personas se sintiesen más cómodas al integrarse mejor con sus sistemas.

Otra gran novedad que se podría incluir es el histórico de las estadísticas. Se podría ofrecer a los usuarios un histórico sobre la evolución del rendimiento en las zonas de modo que se pueden ver las zonas que han experimentado un auge o un declive. Esta característica se tendría que implementar con los años a medida que se van guardando datos del pasado ya que actualmente no hay ningún servicio que ofrezca datos de inmuebles pasados.

También se podría crear un sistema de notificaciones, avisos y subscripciones de modo que los usuarios se pudieran subscribir para enterarse de cambios importantes en las zonas que estén interesados en invertir o ver su evolución.

Otra característica que ayudaría a la aplicación a ofrecer unos mejores datos sería la inclusión de más fuentes de información como la consulta de más inmuebles en diferentes portales de compra/venta de inmuebles.

Una característica que ayudaría mucho a las empresas que se dedican a la inversión que serían atraídas a la aplicación sería desarrollar una aplicación de escritorio que aportase más datos sobre las inmuebles de manera que la puedan usar para mejorar sus inversiones o aconsejar mejor a terceros.

Bibliografía

[1] "Real estate activity statistics - nace rev. 2." https://ec.europa.eu/eurostat/statistics-explained/index.php/Real_estate_activity_statistics_-_NACE_Rev._2#Structural_profile.

[2] P. A. y Alberto Urtasun, "Evolución reciente del mercado de la vivienda en españa," *Banco de España*. (Descargar).

[3] Redacción, "El mercado de la vivienda movió 74.000 millones en 2017, la cifra más alta en siete años," *Idealista news*. (Descargar).

[4] C. Ruiz, "Esto es lo que le espera al mercado inmobiliario en 2019," *El Mundo*. (Descargar).

[5] C. Ruiz, "El sector inmobiliario se mueve, pero "sigue a años luz"de la época del 'boom'," *El Mundo*. (Descargar).

[6] A. Pascual, "Nuevo 'boom' de las inmobiliarias: El sector es una selva de comisionistas analfabetos," *El Confidencial*. (Descargar).

[7] R. Vallés, "La compra como inversión: la tendencia que se consolida en el sector inmobiliario," *La Vanguardia*. (Descargar).

[8] "¿qué es la inversión?," *BBVA*. (Descargar).

[9] "Idealista data." https://www.idealista.com/data/.

[10] "Idealista news." https://www.idealista.com/news/.

[11] Redacción, "La rentabilidad de la inversión de vivienda se reduce hasta el 7,4 % en el tercer trimestre," *Idealista*, 16/10/2018. (Descargar).

[12] Redacción, "¿en qué zona de españa es más rentable comprar piso? ¿y alquilar? (tablas)," *Idealista*, 16/10/2018. (Descargar).

[13] "Descuentos por municipios pedidos por los compradores de vivienda." https://www.idealista.com/news/estadisticas/descuentos-vivienda/venta-viviendas/municipios.

[14] E. Sanz, "Los barrios más rentables para alquilar un piso también son los más arriesgados," *El Confidencial*, 26/06/2018. (Descargar).

[15] L. Julbe, "¿en qué ciudades es más rentable invertir en la compra de una vivienda?," *El Economista*, 10/02/2020. (Descargar).

[16] Redacción, "La vivienda puede rentar hasta un 7 %," *Expansión*, 04/08/2019. (Descargar).

[17] "Idealista." https://www.idealista.com.

[18] "Evolución del precio de la vivienda en venta en españa." https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/.

[19] "Fotocasa." https://www.fotocasa.es/.

[20] Redacción, "¿en qué lugar de españa es más rentable la vivienda?," *Fotocasa*, 10/02/2020. (Descargar).

[21] Redacción, "La rentabilidad de la vivienda en españa se sitúa en 6,6 % en 2019, el dato más elevado de los últimos 14 años," *Fotocasa*, 10/02/2020. (Descargar).

[22] "Django." https://www.djangoproject.com/.

[23] "Python." https://www.python.org/.

[24] "Postgresql." https://www.postgresql.org/.

[25] "Pgadmin." https://www.pgadmin.org/.

[26] "Bulma css." https://bulma.io/.

[27] "Geopy." https://geopy.readthedocs.io/en/stable/.

[28] "Nominatim." https://nominatim.org/.

[29] "Charts.js." https://www.chartjs.org/.

[30] "Google maps." https://developers.google.com/maps/documentation/javascript/tutorial/.

[31] "Idealista labs." https://www.idealista.com/labs/.

[32] "Qué son las api y para qué sirven." https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces.

[33] "Rfc 1738." https://www.ietf.org/rfc/rfc1738.txt.

[34] "Requests." https://requests.readthedocs.io/en/master/.

[35] "Requests outhlib." https://github.com/requests/requests-oauthlib.

[36] "Scrapy." https://scrapy.org/.

[37] "Xpath." https://www.w3.org/TR/1999/REC-xpath-19991116/.

[38] Ginopalazzo, "Dedomeno." https://github.com/ginopalazzo/dedomeno.

[39] "Scrapy-djangoitem." https://github.com/scrapy-plugins/scrapy-djangoitem.

[40] "Scrapy-user-agents." https://github.com/hyan15/crawler-demo/tree/master/crawling-basic/scrapy_user_agents.

[41] "Scrapy proxies." https://github.com/aivarsk/scrapy-proxies.

Apéndices

A

Modelos de la base de datos en Django

En los modelos de Django, los campos **ChaField** son campos de texto de longitud variable hasta un máximo de caracteres indicado por el parámetro **max_lenght**. En el caso de que la opción **null** sea **True** indica que ese campo puede contener valores nulos. El parámetro **default** indica el valor que tendrá por defecto ese campo. Los campos **BooleanField** son campos que pueden contener valores booleanos. El campo **URLField** se guarda en la base de datos como un valor de texto normal, pero Django se asegura de que el valor que contiene es una URL válida. Los campos **PositiveInteger** y **SmallPositiveInteger** guardan valores enteros sin signo de 32bits y de 16bits respectivamente. Los campos **FloatField** son campos que pueden guardar números en coma flotante. El parámetro **choices** sirve para indicar los valores que puede tomar ese campo mediante una lista o enumeración.

A.1. Modelos para la API de Idealista

Código A.1: Modelo para guardar propiedades de la API de Idealista

```
class ApiProperty(models.Model):
    property_code = models.PositiveSmallIntegerField(primary_key=True, editable=False)
    title = models.CharField(max_length=500)
    subtitle = models.CharField(max_length=500)

    latitude = models.FloatField(db_index=True)
    longitude = models.FloatField(db_index=True)
    distance = models.PositiveIntegerField()

    country = models.CharField(max_length=2, choices=Country.choices, db_index=True)
    province = models.CharField(max_length=100, db_index=True)
    region = models.CharField(null=True, max_length=100)
    subregion = models.CharField(null=True, max_length=100, blank=True, default='')
    municipality = models.CharField(max_length=200, db_index=True)
    district = models.CharField(null=True, max_length=100, db_index=True)
    neighborhood = models.CharField(null=True, max_length=100, db_index=True)
    address = models.CharField(max_length=300)
    floor = models.CharField(null=True, max_length=50)

    size = models.PositiveIntegerField()
    bathrooms = models.PositiveSmallIntegerField()
    exterior = models.BooleanField(default=False)

    state = models.CharField(max_length=1, choices=State.choices)

    has_video = models.BooleanField(default=False)
    num_photos = models.PositiveSmallIntegerField()
    operation = models.CharField(max_length=1, choices=Operation.choices)
    price = models.PositiveIntegerField()

    show_address = models.BooleanField(default=False)

    thumbnail = models.URLField()
    url = models.URLField()
    external_reference = models.CharField(null=True, max_length=100)

    typology = models.CharField(max_length=1, choices=Tipology.choices, default=Tipology.FLAT)
    subtypology = models.CharField(null=True, max_length=2, choices=Subtipology.choices)

    @property
    def price_by_size(self):
        return self.price / self.size
```

A.2. Modelos para datos del *crawler*

Código A.2: Modelo para guardar inmobiliarias mediante los datos extraídos del *crawler de Idealista*

```
class RealEstate(models.Model):
    source = models.CharField(max_length=100)
    source_url = models.URLField(unique=True)
    name = models.CharField(null=True, max_length=500)
    slug = models.SlugField(max_length=500, unique=True)
    logo = models.URLField(null=True)
    url = models.URLField(null=True)
    desc = models.TextField()
    phone = models.CharField(max_length=100)
    address = models.CharField(null=True, max_length=2000)
    country_code = models.CharField(max_length=2, choices=Country.choices)
```

Código A.3: Modelo para guardar propiedades mediante los datos extraídos del *crawler de Idealista*

```
class Property(PolymorphicModel):
    title = models.CharField(null=True, max_length=500)
    source = models.CharField(max_length=20)
    source_url = models.URLField(unique=True)
    source_id = models.BigIntegerField(primary_key=True)
    description = models.TextField()
    operation = models.CharField(max_length=1, choices=Operation.choices)
    type = models.CharField(max_length=1, choices=PropertyType.choices)
    name = models.CharField(null=True, max_length=500)
    phone_1 = models.CharField(null=True, max_length=30)
    phone_2 = models.CharField(null=True, max_length=30)
    real_estate = models.ForeignKey(RealEstate, null=True, on_delete=models.SET_NULL,
        related_name='property',
                                    help_text='If blank there is not a real estate involved')
    real_estate_raw = models.CharField(null=True, max_length=200)
    price_raw = models.PositiveIntegerField()


    country_code = models.CharField(max_length=2, choices=Country.choices)
    province = models.CharField(max_length=50)
    region = models.CharField(max_length=100)
    municipality = models.CharField(max_length=100)
    district = models.CharField(null=True, max_length=100)
    neighborhood = models.CharField(null=True, max_length=100)
    address = models.CharField(max_length=1000)
    address_exact = models.BooleanField()
    latitude = models.FloatField()
    longitude = models.FloatField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    online = models.BooleanField(default=True)
```

Código A.4: Modelo para guardar el precio de los inmuebles con el tiempo mediante los datos extraídos del *crawler de Idealista*

```
class Price(models.Model):
    price = models.PositiveIntegerField(editable=False)
    start = models.DateField(auto_now_add=True, editable=False)
    end = models.DateField(null=True)
    property = models.ForeignKey(Property, on_delete=models.CASCADE, editable=False)
```

Código A.5: Modelo para guardar casas mediante los datos extraídos del *crawler de Idealista*

```
class House(Property):
    house_type = models.CharField(max_length=200)
    size = models.PositiveIntegerField()
    used_size = models.PositiveIntegerField(null=True)
    terrain_size = models.PositiveIntegerField(null=True)
    bedrooms = models.PositiveSmallIntegerField()
    bathrooms = models.PositiveSmallIntegerField()
    orientation = models.CharField(null=True, max_length=2, choices=Orientation.choices)
    state = models.CharField(max_length=1, choices=State.choices)
    construction_year = models.PositiveIntegerField(null=True)
    has_garage = models.BooleanField(default=False)
    terrace = models.BooleanField(default=False)
    chimney = models.BooleanField(default=False)
    store_room = models.BooleanField(default=False)
    builtin_wardrobes = models.BooleanField(default=False)
    clothesline_space = models.BooleanField(default=False)
    furnished = models.BooleanField(default=False)
    furnished_kitchen = models.BooleanField(default=False)

    floor_num = models.CharField(null=True, max_length=50)
    position = models.CharField(null=True, max_length=1, choices=Position.choices)
    elevator = models.BooleanField(default=False)

    garden = models.BooleanField(default=False)
    air_conditioning = models.BooleanField(default=False)
    swimming_pool = models.BooleanField(default=False)
```

Código A.6: Modelo para guardar habitaciones mediante los datos extraídos del *crawler de Idealista*

```
class Room(Property):
    house_type = models.CharField(max_length=100)
    size = models.PositiveIntegerField()
    floor_num = models.CharField(null=True, max_length=200)
    elevator = models.BooleanField(default=False)
    bed_type = models.CharField(max_length=1, choices=BedType.choices)
    bedrooms = models.PositiveSmallIntegerField()
    bathrooms = models.PositiveSmallIntegerField()
    min_month_stay = models.PositiveSmallIntegerField(default=0)
    max_people = models.PositiveSmallIntegerField(null=True)
    people_now_living_gender = models.CharField(max_length=1, choices=Gender.choices)
    people_now_living_age_min = models.PositiveSmallIntegerField(null=True)
    people_now_living_age_max = models.PositiveSmallIntegerField(null=True)
    smoking_allowed = models.BooleanField(default=False)
    pet_allowed = models.BooleanField(default=False)

    looking_for_gender = models.CharField(max_length=1, choices=Gender.choices)
    couples_allowed = models.BooleanField(default=False)
    looking_for_student = models.BooleanField(default=False)
    looking_for_worker = models.BooleanField(default=False)
    gay_friendly = models.BooleanField(default=False)

    air_conditioning = models.BooleanField(default=False)
    internet = models.BooleanField(default=False)
    builtin_wardrobes = models.BooleanField(default=False)
    furnished = models.BooleanField(default=False)
    house_cleaners = models.BooleanField(default=False)
```

Código A.7: Modelo para guardar oficinas mediante los datos extraídos del *crawler de Idealista*

```
class Office(Property):
    size = models.PositiveIntegerField()
    used_size = models.PositiveIntegerField(null=True)
    terrain_size = models.PositiveIntegerField(null=True)
    num_of_floors = models.IntegerField(null=True)
    distribution = models.CharField(null=True, max_length=200)
    kitchen = models.BooleanField(default=False)
    bathrooms = models.PositiveSmallIntegerField(default=0)
    bathrooms_location = models.CharField(null=True, max_length=1,
        choices=BathroomLocation.choices)
    orientation = models.CharField(max_length=2, choices=Orientation.choices)
    state = models.CharField(max_length=1, choices=State.choices)
    has_garage = models.BooleanField(default=False)

    floor_num = models.CharField(null=True, max_length=200)
    position = models.CharField(null=True, max_length=1, choices=Position.choices)
    elevators = models.PositiveSmallIntegerField(default=0)
    office_building_use = models.CharField(null=True, max_length=20)
    janitor = models.BooleanField(default=False)
    access_control = models.BooleanField(default=False)
    security_system = models.BooleanField(default=False)
    security_door = models.BooleanField(default=False)
    fire_extinguishers = models.BooleanField(default=False)
    fire_detectors = models.BooleanField(default=False)
    sprinklers = models.BooleanField(default=False)
    fire_door = models.BooleanField(default=False)
    emergency_exit = models.BooleanField(default=False)
    emergency_exit_lights = models.BooleanField(default=False)

    store_room = models.BooleanField(default=False)
    hot_water = models.BooleanField(default=False)
    heating = models.BooleanField(default=False)
    air_conditioning_cold = models.BooleanField(default=False)
    air_conditioning_hot = models.BooleanField(default=False)
    double_glazed_windows = models.BooleanField(default=False)
    false_ceiling = models.BooleanField(default=False)
    false_floor = models.BooleanField(default=False)
```

Código A.8: Modelo para guardar garajes mediante los datos extraídos del *crawler de Idealista*

```
class Garage(Property):
    garage_type = models.CharField(max_length=2, choices=GarageType.choices)
    number = models.PositiveIntegerField(null=True)
    covered = models.BooleanField(default=False)
    elevator = models.BooleanField(default=False)

    automatic_door = models.BooleanField(default=False)
    security_cameras = models.BooleanField(default=False)
    alarm = models.BooleanField(default=False)
    security_guard = models.BooleanField(default=False)
```

Código A.9: Modelo para guardar terrenos mediante los datos extraídos del *crawler de Idealista*

```
class Land(Property):
    size = models.PositiveIntegerField()
    size_min_rent = models.PositiveIntegerField(null=True)
    size_min_sale = models.PositiveIntegerField(null=True)
    size_to_build = models.PositiveIntegerField(null=True)
    access = models.CharField(max_length=1, choices=Access.choices)
    nearest_town = models.CharField(null=True, max_length=200)

    land_type = models.CharField(max_length=1, choices=LandType.choices)
    zoned = models.CharField(null=True, max_length=200)
    building_floors = models.PositiveIntegerField(null=True)

    sewerage = models.BooleanField(default=False)
    street_lighting = models.BooleanField(default=False)
    water = models.BooleanField(default=False)
    electricity = models.BooleanField(default=False)
    sidewalks = models.BooleanField(default=False)
    natural_gas = models.BooleanField(default=False)
```

Código A.10: Modelo para guardar locales mediante los datos extraídos del *crawler de Idealista*

```
class Premise(Property):
    premise_building_type = models.CharField(max_length=1, choices=PremiseBuildingType.choices)
    transfer_price = models.PositiveIntegerField(null=True)
    size = models.PositiveIntegerField()
    used_size = models.PositiveIntegerField(null=True)
    terrain_size = models.PositiveIntegerField(null=True)
    num_of_floors = models.PositiveIntegerField(null=True)
    distribution = models.CharField(null=True, max_length=200)
    location = models.CharField(max_length=1, choices=PremiseLocation.choices)
    corner = models.BooleanField(default=False)
    show_windows = models.PositiveIntegerField(null=True)
    last_activity = models.CharField(null=True, max_length=200)
    state = models.CharField(max_length=1, choices=State.choices)
    bathrooms = models.PositiveSmallIntegerField()

    floor_num = models.CharField(null=True, max_length=200)
    facade = models.CharField(null=True, max_length=200)

    air_conditioning = models.BooleanField(default=False)
    alarm_system = models.BooleanField(default=False)
    store_room = models.BooleanField(default=False)
    heating = models.BooleanField(default=False)
    kitchen = models.BooleanField(default=False)
    security_door = models.BooleanField(default=False)
    smoke_extractor = models.BooleanField(default=False)
```

Código A.11: Modelo para guardar trasteros mediante los datos extraídos del *crawler de Idealista*

```
class StoreRoom(Property):
    size = models.PositiveIntegerField()
    height = models.PositiveSmallIntegerField(null=True)
    access_24h = models.BooleanField(default=False)
    limited_parking = models.BooleanField(default=False)
```

Código A.12: Modelo para guardar edificios mediante los datos extraídos del *crawler de Idealista*

```
class Building(Property):
    size = models.IntegerField()
    size_min_rent = models.IntegerField(null=True)
    building_type = models.CharField(null=True, max_length=200)
    elevators = models.PositiveIntegerField(default=0)
    floor_num = models.PositiveIntegerField(null=True)
    garage_num = models.PositiveIntegerField(null=True)
    security = models.BooleanField(default=False)
    state = models.CharField(max_length=1, choices=State.choices)
    tenant = models.BooleanField(null=True)
    house_num = models.PositiveIntegerField(null=True)
    construction_year = models.PositiveSmallIntegerField(null=True)
```

B

Enumeraciones de los modelos de la base de datos en Django

Código B.1: Opciones que puede tomar el tipo de una propiedad

```
class PropertyType(models.TextChoices):
    GARAGE = 'G', _("Garages")
    HOUSE = 'H', _("House")
    OFFICE = 'O', _("Offices")
    PREMISE = 'P', _("Premises")
    LAND = 'L', _("Land")
    ROOM = 'R', _("Room")
    STOREROOM = 'S', _("Storeroom")
    BUILDING = 'B', _("Building")
```

Código B.2: Opciones que puede tomar el país de una propiedad

```
class Country(models.TextChoices):
    SPAIN = 'ES', _("Spain")
    ITALY = 'IT', _("Italy")
    PORTUGAL = 'PT', _("Portugal")
```

Código B.3: Opciones que puede tomar la operación de una propiedad

```
class Operation(models.TextChoices):
    SALE = 'S', _("Sale")
    RENT = 'R', _("Rent")
```

Código B.4: Opciones que puede tomar el estado del inmueble

```
class State(models.TextChoices):
    EMPTY = '', ''
    NEW_DEVELOPMENT = 'N', _("New development")
    GOOD = 'G', _("Good condition")
    RENEW = 'R', _("Needs renovating")
```

Código B.5: Opciones que puede tomar la tipología de inmueble

```
class Tipology(models.TextChoices):
    CHALET = 'L', _("Chalet")
    COUNTRY_HOUSE = 'C', _("Country house")
    FLAT = 'F', _("Flat")
    GARAGE = 'G', _("Garage")
    OFFICE = 'O', _("Office")
    PREMISE = 'P', _("Premise")
    ROOM = 'R', _("Room")
```

Código B.6: Opciones que puede tomar la subtipología de inmueble

```
class Subtipology(models.TextChoices):
    DUPLEX = 'DP', _("Duplex")
    PENTHOUSE = 'PH', _("Penthouse")
    STUDIO = 'ST', _("Studio")

    INDEPENDANT_HOUSE = 'IH', _("Independant house")
    SEMIDETACHED_HOUSE = 'MH', _("Semidetached house")
    TERRACED_HOUSE = 'TH', _("Terraced house")
    ANDAR_MORADIA = 'AM', _("Andar moradia")

    COUNTRY_HOUSE = 'CH', _("Country house")
    CASTILLO = 'CA', _("Castle")
    PALACIO = 'PA', _("Palace")
    MASIA = 'MA', _("Masia")
    CORTIJO = 'CO', _("Cortijo")
    CASALE = 'CS', _("Casale")
    CASA_DE_PUEBLO = 'PU', _("Smalltown house")
    CASA_TERRERA = 'CT', _("Casa terrera")
    CASA_MATA = 'MT', _("Casa mata")
    TORRE = 'TO', _("Tower")
    CASERON = 'CE', _("Caseron")
    PAZO = 'PZ', _("Pazo")
    VILLA = 'VI', _("Village")
    PALACETE = 'PE', _("Palacete")
    MASSERIA = 'MS', _("Masseria")
    FATTORIA = 'FT', _("Fattoria")

    TRULLO = 'TR', _("Trullo")
    CASALICASCINE = 'CI', _("Casalicascine")
    BAITE_CHALET = 'BC', _("Baite chalet")
    QUINTA = 'QU', _("Quinta")
    MOINHO = 'NH', _("Moinho")
    MONTE_ALENTEJANO = 'AL', _("Alentejano mount")
    SOLAR = 'SO', _("Plots")

    COMMERCIAL_PROPERTY = 'CP', _("Commercial property")
    INDUSTRIAL_PREMISE = 'IP', _("Industrial premise")

    CAR_AND_MOTORCYCLE = 'CM', _("Car and motorcycle")
    COMPACT_CAR = 'CC', _("Compact car")
    SEDAN_CAR = 'SC', _("Sedan car")
    MOTORCYCLE = 'MO', _("Motorcycle")
    TWO_CARS = '2C', _("Two cars")
```

Código B.7: Opciones que puede tomar la orientación de una propiedad

```
class Orientation(models.TextChoices):
    NORTH = 'N', _("North")
    SOUTH = 'S', _("South")
    EAST = 'E', _("East")
    WEST = 'W', _("West")
    NORTHEAST = 'NE', _("Northeast")
    SOUTHEAST = 'SE', _("Southeast")
    SOUTHWEST = 'SW', _("Southwest")
    NORTHWEST = 'NW', _("Northwest")
```

Código B.8: Opciones que puede tomar el género de una persona

```
class Gender(models.TextChoices):
    GIRLS = 'G', _("Girl(s)")
    BOYS = 'B', _("Boy(s)")
    MIXED = 'X', _("Mixed")
```

Código B.9: Opciones que puede tomar el tipo de una cama

```
class BedType(models.TextChoices):
    INDIVIDUAL = 'I', _("Individual␣bed")
    DOUBLE = 'D', _("Double")
    DUAL = '2', _("Two␣beds")
    WITHOUT = 'N', _("Without␣bed")
```

Código B.10: Opciones que puede tomar la posición de una propiedad

```
class Position(models.TextChoices):
    INTERIOR = 'I', _("Interior")
    EXTERIOR = 'E', _("Exterior")
```

Código B.11: Opciones que puede tomar la posición de un baño

```
class BathroomLocation(models.TextChoices):
    INSIDE = 'I', _("Inside")
    OUTSIDE = 'E', _("Outside")
```

Código B.12: Opciones que puede tomar el tipo de un garaje

```
class GarageType(models.TextChoices):
    CAR_AND_MOTORCYCLE = 'CM', _("Car and motorcycle")
    COMPACT_CAR = 'CC', _("Compact car")
    SEDAN_CAR = 'SC', _("Sedan car")
    MOTORCYCLE = 'MO', _("Motorcycle")
    TWO_CARS = '2C', _("Two cars")
```

Código B.13: Opciones que puede tomar el diseño de una oficina

```
class OfficeLayout(models.TextChoices):
    WITH_WALLS = 'W', _("With walls")
    OPEN_PLAN = 'O', _("Open plan")
x
```

Código B.14: Opciones que puede tomar el uso del edificio de una oficina

```
class OfficeBuildingUse(models.TextChoices):
    EXCLUSIVE = 'X', _("Exclusive")
    MIXED = 'M', _("Mixed")
```

Código B.15: Opciones que puede tomar el acceso de un terreno

```
class Access(models.TextChoices):
    HIGHWAY = 'H', _("Highway")
    ROAD = 'R', _("Road")
    STREET = 'S', _("Street")
```

Código B.16: Opciones que puede tomar el tipo de un terreno

```
class LandType(models.TextChoices):
    URBAN = 'U', _("Urban")
    DEVELOPABLE = 'D', _("Developable")
    UNDEVELOPED = 'N', _("Undeveloped")
```

Código B.17: Opciones que puede tomar la localización de un local

```
class PremiseLocation(models.TextChoices):
    SHOPPING_CENTER = 'C', _("In a shopping center")
    STREET = 'S', _("On street level")
    MEZZANINE = 'M', _("Mezzanine")
    UNDERGROUND = 'U', _("Underground")
    OTHERS = 'O', _("Others")
```

Código B.18: Opciones que puede tomar el tipo del edificio de un local

```
class PremiseBuildingType(models.TextChoices):
    PREMISES = 'P', _("Premises")
    INDUSTRIAL_BUILDING = 'I', _("Industrial building")
```

C

Interfaz de los filtros de búsqueda

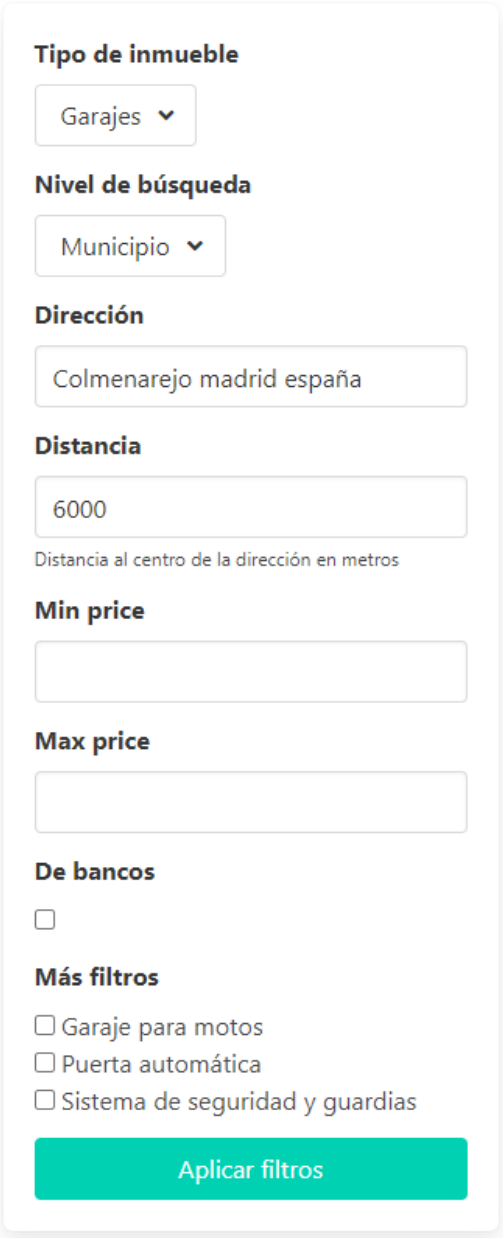

Figura C.1: Filtro de garajes de la aplicación

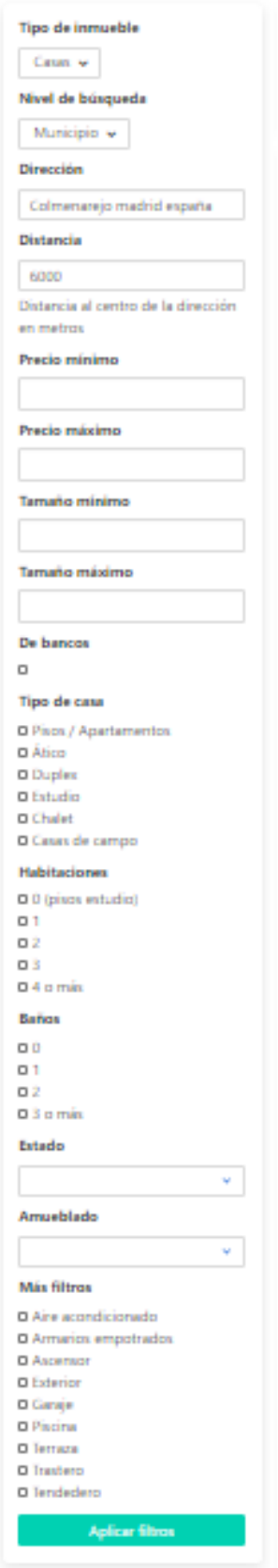

Figura C.2: Filtro de viviendas de la aplicación

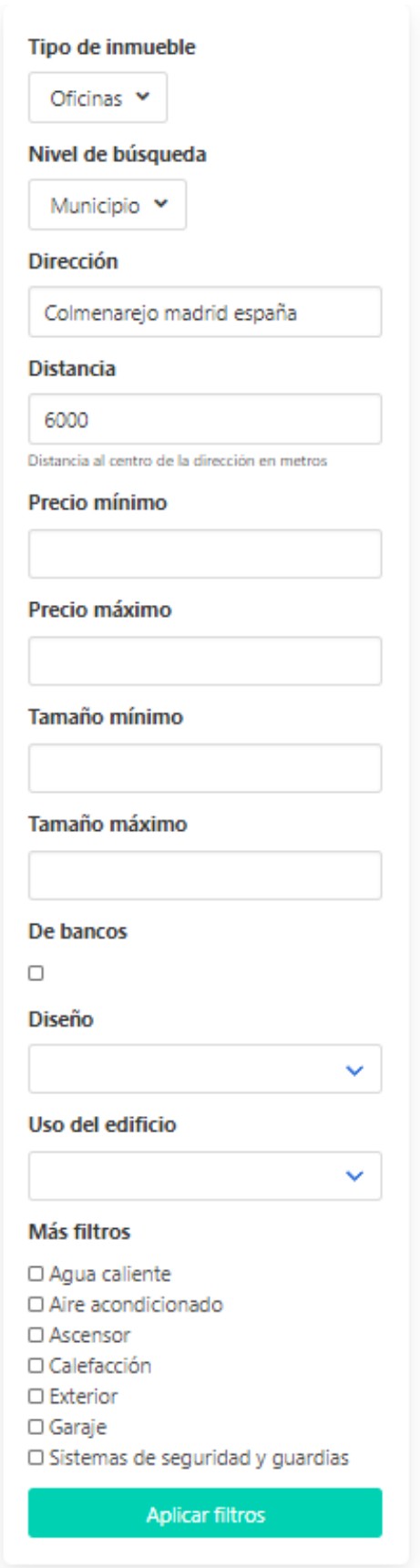

Figura C.3: Filtro de oficinas de la aplicación

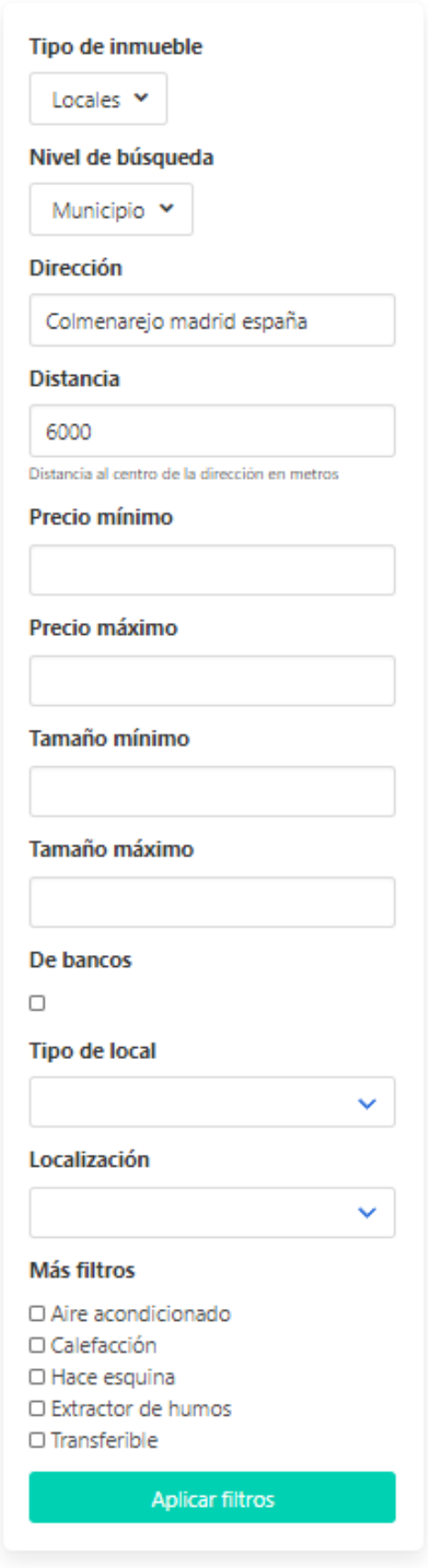

Figura C.4: Filtro de locales de la aplicación

D

Ejemplos de formularios en Django

Código D.1: Ejemplo de formulario en Django

```
class PropertyAdvancedRecomendationForm(forms.Form):
    address = forms.CharField(label=_("Address"), min_length=4, max_length=100,
        widget=forms.TextInput(attrs={'class': 'input'}))
    distance = forms.IntegerField(label=_("Distance"), min_value=500, max_value=10000000,
                                  help_text=_("Distance to the center of the adddres in meters"),
                                  widget=forms.NumberInput(attrs={'class': 'input'}))
    min_price = forms.FloatField(label=_("Min price"), min_value=10000, max_value=100000000,
        required=False,
                                 widget=forms.NumberInput(attrs={'class': 'input'}))
    max_price = forms.FloatField(label=_("Max price"), min_value=10000, max_value=100000000,
        required=False,
                                 widget=forms.NumberInput(attrs={'class': 'input'}))
    min_size = forms.IntegerField(label=_("Min size"), min_value=60, max_value=1000, required=False,
                                  widget=forms.NumberInput(attrs={'class': 'input'}))
    max_size = forms.IntegerField(label=_("Max size"), min_value=60, max_value=1000,
        required=False,
                                  widget=forms.NumberInput(attrs={'class': 'input'}))
    bank_offer = forms.BooleanField(label=_("Bank offer"), required=False)

class HomePropertyAdvancedRecomendationForm(PropertyAdvancedRecomendationForm):
    select_fields = ('state', 'furnished')
    home_types = forms.MultipleChoiceField(label=_("Type of home"), choices=HomeType.choices,
        widget=form .CheckboxSelectMultiple, required=False)
    bedrooms = forms.MultipleChoiceField(label=_("Bedrooms"), choices=HomeBedroom.choices,
        widget=forms.CheckboxSelectMultiple, required=False)
    bathrooms = forms.MultipleChoiceField(label=_("Bathrooms"), choices=HomeBathroom.choices,
        widget=forms.CheckboxSelectMultiple, required=False)
    state = forms.ChoiceField(label=_("State"), choices=State.choices, required=False)
    furnished = forms.ChoiceField(label=_("Furnished"), choices=HomeFurnished.choices, required=False)
    more_filters = forms.MultipleChoiceField(label=_("More filters"), choices=HomeMoreFilters.choices,
        widget=forms.CheckboxSelectMultiple, required=False)

class PremisePropertyAdvancedRecomendationForm(PropertyAdvancedRecomendationForm):
    select_fields = ('premise_type', 'location')
    premise_type = forms.ChoiceField(label=_("Type of premise"),
        choices=PremiseBuildingType.choices, required=False)
    location = forms.ChoiceField(label=_("Location"), choices=PremiseLocation.choices, required=False)
    more_filters = forms.MultipleChoiceField(label=_("More filters"),
        choices=PremiseMoreFilters.choices, widget=forms.CheckboxSelectMultiple, required=False)

class OfficePropertyAdvancedRecomendationForm(PropertyAdvancedRecomendationForm):
    select_fields = ('layout', 'building_use')
    layout = forms.ChoiceField(label=_("Layout"), choices=OfficeLayout.choices, required=False)
    building_use = forms.ChoiceField(label=_("Building use"), choices=OfficeBuildingUse.choices,
        required=False)
    more_filters = forms.MultipleChoiceField(label=_("More filters"), choices=OfficeMoreFilters.choices,
        widget=forms.CheckboxSelectMultiple, required=False)
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID